The University of Osaka
Institutional Knowledge Archive

Title	高信頼通信ソフトウェア構成に関する研究
Author(s)	小柳, 恵一
Citation	大阪大学, 1998, 博士論文
Version Type	VoR
URL	https://doi.org/10.11501/3144019
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

高信頼通信ソフトウェア構成に関する研究

大阪大学大学院 工学研究科 通信工学専攻

1997年12月

小 柳 恵 一

高信頼通信ソフトウェア構成に関する研究

大阪大学大学院 工学研究科 通信工学専攻

1997年12月

小柳恵一

内容梗概

高度情報化社会に向けて、次世代の通信網の実現に関わる研究が盛んに進められている。この情報化社会に関連する産業の裾野は広範囲に広がると期待されているが、それを支えるインフラストラクチャとして、通信ネットワークの高度化は重要な課題である。

その基盤となるのは、高度なシステムアーキテクチャとそれを実現する、超高集積半導体回路技術、超高速ディジタル回路技術に加えてマルチメディアに代表されるインテリジェントなサービスを実現するソフトウェア技術などである。

既に、通信システムでは、ソフトウェア技術が大きな役割を果たしており、今後ますますその重要性は高まってくる。大規模化していく通信ソフトウェアを従来にも増して高信頼性を実現しつつ、より高度なサービスをユーザに早期に提供することが今後さらに大切になって来るものと考えられる。

即ち、リアルタイム性と高信頼性を追求してきた通信ソフトウェアに対して、これを踏まえて更にサービスの追加・変更に対する即応性とネットワークワイドな運用性の向上が益々重要になってきている。

これらの要望を同時に実現するための高信頼通信ソフトウェア構成に関する研究が、今まで十分に進められてきたとはいいがたい。サービスや機能の追加は年に 1～2 回程度実施されるシステムファイル更新に依らざるを得ないのが現状である。

本論文では、上述した要望に応えるために、ユーザに影響を与えることなく、随時運用中の通信システムに新たな（複数の）オブジェクトのプラグイン及びそのプラグアウトを可能とする高信頼通信ソフトウェア構成に関する研究の成果をまとめたものであり、次の 7 章から構成される。

第 1 章で近年の通信ソフトウェアへの要求条件を論述している。そして IT (Information Technology) 分野などで提案されている先端技術ではこれらの要望に応えることが出来ないことを明らかにしている。それに応えるための技術提案が本研究の狙いであり、それらの要求条件に応えるためには 5 つの課題、即ち、①プログラムの着脱単位を明確にする通信ソフトウェア構成技術、②プラグイン・アウト構築技術、③その機能を多様な通信システムに適用するためのプラットフォーム構築技術と④システム・ネットワーク管理技術、及び⑤オブジェクト管理を容易にするための通信ソ

フトウェア開発支援技術、を解決する必要があることを明らかにしている。第 2 章以降は，これらの課題に対応した技術毎に章を構成している。第 2 章では通信システムのモデル化と階層化技術手法を実現するためのオブジェクト指向通信ソフトウェア構成技術を明らかにしている。これによりオブジェクト単位のサービス・機能追加の着脱が容易となり、次の章で紹介するプラグイン・アウトとの親和性が図れることを示しいている。第 3 章では，そのオブジェクト単位に着脱を可能にするプラグイン・アウト技術を明らかにしている。リアルタイム性と高信頼性を踏まえた上でオブジェクトの追加･変更を可能とするための技術として運用中のオブジェクトを置き換えるための機構と、当初予期していなかった新規オブジェクトを追加するためのプラグイン機構を提案している。これにより，従来のシステムファイルによる立ち上げをまたずにサービス・機能の追加ができることから，今後の通信システム構築における必須機能である。第 4 章では，この様なプラグイン・アウト機能を多種多様な通信システムに適用するための共通プラットフォーム化技術を明らかにしている。即ち、プラグイン・アウト機能の適用範囲を拡大するために第 2 章で提案したオブジェクト指向技術が提供する機能継承により、全システムに共通なコア機能に手を入れずに、各システム毎に依存するオプション機能を追加する技術を提案している。これによりプラットフォームを利用するユーザ側のシステムデバッグ工数などの削減が可能となる。更に，システムの分散化をサービスプログラム設計者に意識させない言語レベルの分散 OS 技術を明らかにしている。これによりサービスプログラム設計者がサービスのみを意識すればよい方式が提供できる。第 5 章では，上記機能を実現する多様な通信システムとその通信システムからなるネットワークの管理を可能とするために ITU-T で標準化が進められている TMN（Telecommunication Management Network）制御の適用技術を明らかにしている。分散化されたシステムへの TMN 適用に当たって、応答時間の評価からシステム間プロトコルプロファイルを明確にすると共にその実装技術を提案している。これによりシステムおよびネットワーク管理者による容易なプラグイン・アウトなどのファイル管理が実現可能となる。第 6 章は，本研究で提案するオブジェクト指向通信ソフトウェアの開発に向けた開発支援技術を明らかにしている。論理的なリソースをサービス階層に提供可能な通信ソフトウェア構成で、呼処理などのサービスプログラムを形式的な仕様記述に依り表現可能であることを宣言的実行可能な言語である Prolog により示し、その可能性を明確にしている。最後の第 7 章では本研究で提案する高信頼通信ソフトウェア構築技術を総括している。

もくじ

謝辞 ____ *1*

第1章 序論 ____ *2*

1.1 背景と目的 ____ 2

1.2 要求条件と課題 ____ 4

1.3 従来研究と本論文の関係 ____ 6

1.4 論文の構成と各章の概要 ____ 11

第2章 通信ソフトウェア構成技術 ____ *12*

2.1 緒言 ____ 12

2.2 モデル化と階層化手法 ____ 12

2.3 通信ソフトウェア設計法 ____ 18

2.4 評価 ____ 33

2.5 結言 ____ 36

付録 ____ 38

2.1 通信ソフトウェアの“複雑さ” ____ 38

第3章 プラグイン・アウト構築技術 ____ *41*

3.1 緒言 ____ 41

3.2 課題の明確化 ____ 41

3.2.1 狙い ____ 41

3.2.2 課題の捉えかた ____ 42

3.2.3 システムファイル更新とプラグイン技術 ____ 44

3.3 新オブジェクトのプラグイン ____ 46

3.3.1 新旧ｵﾌﾞｼﾞｪｸﾄの同時走行と共通ﾃﾞｰﾀの引継ぎ ____ 46

3.3.2 新オブジェクトのプラグイン機構 ____ 52

3.4 新規オブジェクトのプラグイン ____ 54

3.4.1 オブジェクト間の呼び出し ____ 54

3.4.2 オブジェクトの初期設定 ____ 56

3.5 プラグアウト機構 ____ 59

3.5.1 新オブジェクトのプラグアウト ____ 59

3.5.2 新規オブジェクトのプラグアウト ____ 60

3.6 オブジェクトのファイル化 ____ 61

3.7 評価 ____ 62

3.7.1 静的評価 ____ 62

3.7.2 動的評価 ____ 64

3.8 結言 ____ 68

付録 ____ 70

3.1 プラグイン時のファイル化単位の工夫 ____ 70

3.2 中断があるソフトウェアの対策 73
第4章 プラットフォーム構築技術 78
4.1 緒言 78
4.2 プラットフォーム構築 78
4.2.1 階層化 78
4.2.2 オブジェクトモデル 80
4.3 カスタマイズ化 82
4.4 分散制御プラットフォーム 85
4.4.1 オブジェクトモデル 86
4.4.2 言語モデル 88
4.4.3 障害透過性と呼救済 92
4.5 評価 93
4.6 結言 95
第5章 システム及びネットワーク管理技術 96
5.1 緒言 96
5.2 ネットワーク管理制御 96
5.3 評価 99
5.3.1 プロトコルプロファイル 100
5.3.2 コマンド応答時間評価式と数値 103
5.4 結言 105
第6章 通信ソフトウェア開発支援技術 106
6.1 緒言 106
6.2 流通と流用 106
6.3 部品化とライブラリ 109
6.4 評価 111
6.4.1 サービスプログラムの自動生成 111
6.5 結言 115
第7章 結論 117
7.1 オブジェクト指向通信ソフトウェア構成技術の提案 117
7.2 プラグイン・アウト構築技術の提案 118
7.3 共通プラットフォーム構築技術の提案 118
7.4 システム及びネットワーク管理技術の提案 119
7.5 開発支援環境技術の提案 119
参考文献 121
発表論文 123

謝辞

本論文執筆に関して懇切なご指導、ご鞭撻を賜った大阪大学大学院工学研究科教授池田博昌博士に心からの深謝の意を表します。

また、その成果をまとめるにあたり大阪大学大学院工学研究科教授元田浩博士には、終始懇切丁重なご指導とご鞭撻を賜ったことに謹んで感謝の意を表します。

本論文に対して有益なご討論、ご助言を頂いた大阪大学大学院工学研究科教授森永規彦博士、小牧省三博士、前田肇博士、児玉祐治博士、ならびに、長谷川晃博士に謹んで深謝の意を表わします。

本論文に関してご教示、ご指導を賜った大阪大学大学院工学研究科通信工学専攻助教授山本幹博士、ならびに、助手戸出英樹博士に深謝の意を表します。

また、常々ご指導頂き本論文執筆の機会を与えて頂いたトーキン(株)取締役土屋治彦博士（元 NTT 通信網総合研究所主席研究員)、NTT 通信網総合研究所所長鈴木滋彦博士、NTT 技術開発センタ山口治男博士，NTT ヨーロッパ社長富田修二博士に心から感謝致します。

本研究は、筆者が NTT ネットワークサービスシステム研究所において、1986 年から行ったものであり、上司として本研究を行う機会を与えて頂いた文部省国文学研究資料館教授丸山勝巳博士、NTT ソフトウェア門田充弘部長に深く感謝致します。

また、研究遂行にあたり、日々、有益なご討論、ご助言を頂いた、甲斐俊洋、稲守久由、須永宏、渡部信幸、山田徹靖、上田清志、岡本明、松村一、林隆茂をはじめ、通信網総合研究所の方々に心から感謝申し上げます。

第1章序論

【関連学術論文【A2】【A3】【A4】【A5】【A6】】

本章では、本論文に関連する研究分野について述べ、本研究の背景と目的、要求条件と課題、従来研究との関係ならびに本論文の構成について言及する。

1.1 背景と目的

高度情報化社会に向けて、次世代の通信網の実現に関わる研究が盛んに進められている。この情報化社会に関連する産業の裾野は広範囲に広がると期待されているが、それを支えるインフラストラクチャとして、通信ネットワークの高度化は重要な課題である。

その基盤となるのは、高度なシステムアーキテクチャとそれを実現する、超高集積半導体回路技術、超高速ディジタル回路技術に加えてマルチメディアに代表されるインテリジェントなサービスを実現するソフトウェア技術などである。

既に、通信システムでは、ソフトウェア技術が大きな役割を果たしており、今後ますますその重要性は高まってくる。大規模化していく通信ソフトウェアを従来にも増して高信頼性を実現しつつ、より高度なサービスをユーザに早期に提供することが今後さらに大切になって来るものと思われる。

このような通信網技術とソフトウェア技術のさらなる融合（図 1）は、これからの高度情報化社会を形成するための重要な鍵の一つである。リアルタイム性と高信頼性を追求してきた通信ソフトウェアに対して、これを踏まえて更にサービスの追加・変更に対する即応性とネットワークワイドな運用性の向上が益々重要になってきている。

これらの要望を同時に実現するための高信頼通信ソフトウェア構成に関する研究が、今まで十分に進められてきたとはいいがたい。運用中システムへのサービス追加が通信中のユーザに影響を与えるシステムファイル更新によらざるを得ないのが現状である。これに応えるためには仕様化技術から開発環境、そしてソフトウェア構成技術からネットワーク・システム管理技術などに関わるソフトウェア技術に適用可能な一貫したソフトウェア設計法とそれに対応したソフトウェア技術が重要である。その１つの提案としてオブジェクト指向設計法を基本とした技術が提案されている。

しかし、単にオブジェクト指向に基づき通信ソフトウェアを設計・構築しても前述の目的を達成できるものではない。対象とする問題を解決する技術を伴わない通信ソフトウェア設計技術は非生産的かつ非効率な産物をもたらす結果となる。

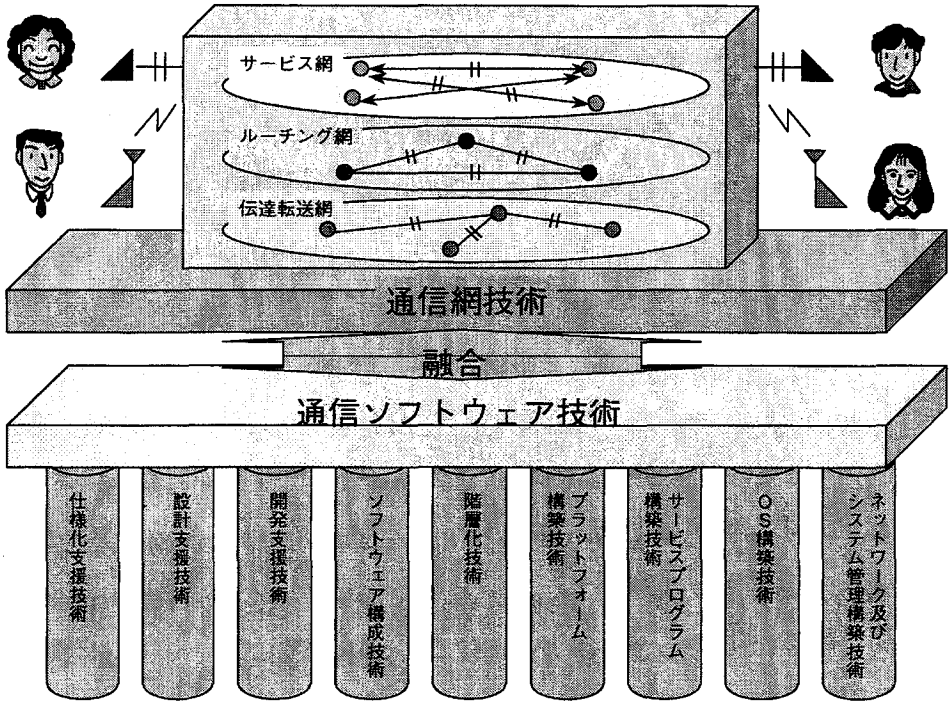


図 1　通信網技術とソフトウェア技術

筆者は、前述の要望に応える為に、ユーザに影響を与えることなく運用中の通信システムに新たな（複数の）オブジェクトを追加変更可能な通信ソフトウェア技術、ソフトウェア維持管理を容易にするオブジェクト指向通信ソフトウェア構成法とそれに対応した開発支援技術など、大規模な高信頼ソフトウェア構成の確立を目指して技術的基本検討から実験とその分析、さらにユーザへの提供に向けた研究・開発を進めてきた。本論文は、その様な検討で得られた成果の中から、高信頼ソフトウェア構成に関する研究をまとめたものである。

また、本論文で提案する技術により、通信ソフトウェアの世界に対して以下の様な効果をもたらすと考える。

(1)通信サービスの設計者は対象とする通信システムの物理的構成を意識しないで済む

(2)通信サービスの設計者は対象とするサービスの実現のみを考慮すれば済む

(3)通信サービスを司るオブジェクトの追加・変更がシステム運用時に随時可能となる

(4)更に、サービスオブジェクトが動作する環境を提供するプラットフォームを構成するオブジェクト自身の追加・変更もある条件化で随時可能となる

(5)従って、定期的に実施されるシステムファイルの更新を待たずにサービス・機能の追加・変更が広範囲に可能となる

(6)また、バグフィックスの場合でも対象とする通信システムに適用されているプロセッサの種別を意識せずに済む（プロセッサフリーである）

(7)この様な環境を提供する通信ソフトウェアプラットフォームを基盤としたシステムと，それにより構築されるネットワークでは、通信ソフトウェア設計者は、対象とする問題をオブジェクトによりモデル化すれば、その抽象化されたオブジェクトの設計のみ考慮すれば済む（オブジェクトのプラグイン・アウト）

1.2 要求条件と課題

リアルタイム性と高信頼性を追求してきた通信ソフトウェアに対して，これを踏まえて更にサービスの追加・変更に対する即応性をネットワークワイドな運用性の向上を考慮したうえで実現するために，通信ソフトウェア技術に要求される条件と課題について本節で述べる。

IT(Information Technology)分野では，Java【1】や ActiveX【2】に代表されるプラグイン技術により NC(Network Computer)，NetPC(NetworkPC)の新たなサービスの創出や従来の PC に関わるソフトウェア維持管理向上を狙いとした新たなネットワーク管理環境【3】の提案がなされている。しかし，近年の IT 分野で発展しているこの様なプラグイン型ソフトウェア技術は，本研究で対象とする高信頼かつリアルタイムな要求の強い交換ソフトウェアの分野などに関わる通信ソフトウェア構築の要求条件を満足できない。

システム内で同時に存在する並列タスク数と要求される応答時間との関係からソフトウェアの領域を３つに分類することができる(図 2)。ソフトウェア構成上，領域Aと領域B/Cとでは，対象とする問題が大きく異なる。

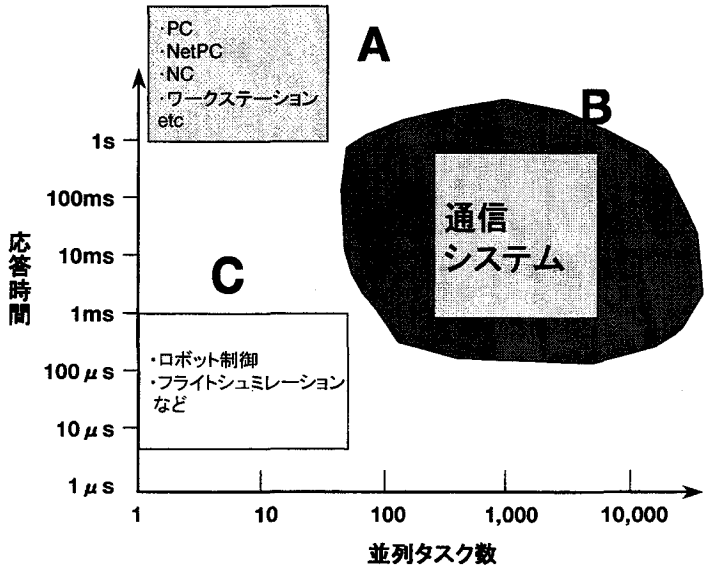


図 2　ソフトウェアの分類—同時並列タスク数と要求される応答時間から見た

上述の Java や ActiveX で代表されるソフトウェア技術は，図 2の領域 A に対応する領域にのみ適応可能である。領域 A に属するシステムでは，基本的にソフトウェアは DISK などの補助メモリに常駐され必要時メインメモリ上に引き上げられ，その際にプログラムが初期設定される。従って，そこに展開されるプラグイン技術は DISK に常駐したプログラムを対象としている。即ち、リアルタイムなシステムへの適用は困難である。また，プラグインの対象となるプログラムのユーザは原則として一人であり，交換システムのサービスプログラムの様に同時に数千のユーザにサービスを提供している様なプログラムを対象としていない。

リアルタイム性と高信頼性を追求する通信ソフトウェア（図 2の領域 B）の設計法とそれに対応したソフトウェア構築技術に向けた要求条件には以下がある。

(1)着脱可能なプログラム単位が明確であること
(2)運用中のシステムにサービスを停止することなく追加・変更が可能なこと

(3)多種多様なプロセッサがシステムに適用されていても，サービス設計者・保守者に意識させることなくサービスの着脱が可能なこと
(4)この様なサービスを提供可能なプラットフォームを各種通信システムに提供可能なこと

この様な要求条件を満足するためには，以下に示す 5 つの通信ソフトウェア技術に関する課題を検討しなければならない。

課題 1： 通信ソフトウェア構成技術の明確化
対象とする通信システムをモデル化し，そのモデルを構成するコンポーネント単位にソフトウェア部品が構成できる技術
課題 2： プラグイン・アウト構築技術の明確化
ソフトウェア部品（オブジェクト）を運用中に着脱可能とする技術
課題 3： プラットフォーム構築技術の明確化
多種多様な通信システムに適用可能とするプラットフォームのカスタマイズ化とシステム構成非依存サービス構築技術
課題 4： システム・ネットワーク管理技術の明確化
この様なソフトウェアプラットフォーム環境からなるシステムおよびネットワークの管理構築技術
課題 5： 通信ソフトウェア開発支援技術の明確化
オブジェクト指向通信ソフトウェア開発支援技術とプラグイン・アウト構築支援技術

1.3 従来研究と本論文の関係

交換システムは、小さなシステムでは数人のユーザを収容するものから大きなシステムでは何十万人ものユーザを収容し、ユーザの要望に応じたサービスを高い信頼性のもとにリアルタイムに提供している。また、ユーザにサービスを提供しながら、保守者によるサービスの追加・変更や交換システムの設備を増設したり減設するための運用サービスを提供している。

この様な交換システムを実現するためには、機能面から捉えると、①ミリ秒オーダの処理が要求される実時間処理、②数万のプロセス制御が必要となる超多重処理、③高信頼性要求と超連続運転、④その連続運転を支える稼動中の増設や機能追加処理、そして⑤通話路などの大量リソース制御技術の確立が必要となる。

従来の交換ソフトウェア技術を、上記 5 つの機能を実現する観点から分類すると、これらの機能を初めてソフトウェア制御により実現し、基本技術を確立した第一世代と、第一世代で問題となったソフトウェア生産性の問題を解決するために、主に高級言語を用いた交換ソフトウェア技術が提案された第二世代に分類することができ

る。しかし、サービス・機能追加技術に着目すると、これら両世代の交換システムは、システム全体のファイルを入れ替えるシステムファイル更新技術に依存せざるを得ないため、近年のサービス・機能追加の即応性に応えることができない。

従来の通信ソフトウェア技術では、システムファイル更新技術でのみサービス・機能追加が可能な理由、そして、本論文が提案するプラグイン・アウトが可能になる理由を以下に示す。

1) 第一世代交換ソフトウェア

交換システム機能を初めてソフトウェア制御により実現した【4】【5】。前述した5つの特徴を持つ機能を満足する交換システムを蓄積プログラム制御方式 により実現している。この時代に交換システムを実現するための基本技術が確立されたことになる。なお，メインメモリ規模が 256kw 程度，プロセッサ処理能力は 0.数 Mips(交換 Mips) の環境下での基本技術の確立であった。この様な制限の下で数万のユーザにサービス提供可能な交換システムを構築する技術が確立された。

確立された基本技術には、実時間処理を実現するための①高･中･低の 3 位レベルのスケジューリング機構、超多重処理を可能とする②特殊トランザクション制御、高信頼性要求と超連続運転を満足するための③冗長構成を用いた障害時自動再立ち上げと救済処理制御、稼動中の増設を可能とする④トリー型データ構造、そして大量リソース制御を実現する⑤局データ・加入者データ管理などがある。

しかし、これら機能を実現する通信ソフトウェアは、全体で1本のファイル構成（分割運用不可能なソフトウェア構造）となっているために、サービス・機能追加は、システムファイル更新に依らざるを得ない方式となっている。

また、交換ソフトウェアの維持管理が問題になり、それを解決するためのプログラムの記述性、保守性の問題が指摘された。IT 分野でも同じ問題が指摘された。それに応えるための技術としてソフトウェア生産性の向上に向けた構造化プログラミング手法【6】や高水準プログラミング言語【7】【8】【9】の研究などが提案された。しかし、これらの技術は、あくまでもプログラム生産性向上に向けた技術提案であり、サービス・機能追加の即応化に応えた技術ではない。

2) 第二世代交換ソフトウェア

蓄積プログラム（SPC）化された交換システムで初めて維持管理面からのソフトウェア構成の見直しが行なわれた。例えば，NTT では，第一世代のソフトウェアをD100A【5】，第二世代をD100B【10】と呼んでいる。また，AT&T では，No. 1ESS【11】が第一世代に，No. 5ESS【12】が第二世代に相当する。No. 4ESS【13】がその過度期的なシステムとして捉えることできる。NTT の場合，ソフトウェア構成としては、D100Bのコンセプトを受け継ぎ、それを交換機制御の面からマルチプロセッサ制御方式に発展させたシステムがD60【14】とD70【15】である。

交換プログラムを機能ブロック化／モジュール化したことによりプログラム全体の見通しが良くなったといえる。例えば，D100B の場合，交換システムを構成する６０個程度の機能ブロック名とそれが実現している機能を知っていれば、機能ブロック名とそれが提供するインタフェース名を理解していることで他の機能ブロック担当の設計者とある程度議論できる。これは，第一世代よりも抽象化された通信モデルが構築されたためだといえる。

しかし、個々の機能ブロックが実現する機能は、第一世代と同じ専用のプロセッサを前提にした作りであり，特殊トランザクション機能などの実行環境を踏襲しているために分割運用不可能なソフトウェア構造となっている。即ち、プログラム記述は分割設計可能であるが、ソフトウェアの運用は一括管理である。そのため、サービス・機能追加は第一世代と同じシステムファイル更新に依らざるを得ない。大規模ソフトウェア構成を機能ブロック化することにより設計時の見通しを良くしたが、サービス・機能を追加するための単位は、第一世代と同様にシステムファイルを一括して更新する手段しか実現されていない。高度インテリジェントネットワーク【16】【17】【18】が提案されたが、高度サービスで、しかもサービスシナリオなどの一部に限定したサービス追加が可能になったに過ぎない。

3) 本研究の狙い

プロセッサ処理能力の向上、メモリチップの低廉化、ワークステーションなどの開発システムの低廉化、高速化、ネットワーク化、ダウンサイジング化などの IT 分野での進歩を十分取り入れる事により、機能やサービス追加の即応化に向けた施策が可能になってきた。通信システムを代表する交換システムが提供するサービスは、第一世代に於けるワイヤードロジックであるクロスバ交換機【19】の単なる置き換えで

はなく、また、第二世代の単なる交換分野に閉じた交換技術者専門家のみが参加できる世界から広く IT 分野の技術を取り入れることにより、より多くの技術者が参加可能で，かつユーザの要望に早急に対応可能なサービス追加性の向上が重要である。

マルチメディア・サービスに代表されるサービスのカスタマイズ化やパーソナル化の実現が必要である。汎用プロセッサや UNIX などの IT 分野の技術を積極的に適用可能な実現機能と、信頼性などの通信システムで積極的に牽引していくべき技術による実現機能とを融合した新たな通信ソフトウェア構成法の確立が益々必要になってきている。

こような通信ソフトウェアへの高度な要求に対して抜本的な改善が要求されている。本研究は、この要望に応えるべき技術を明らかにしている。

具体的には，モジュール化と階層化の徹底である。単に、第二世代のプログラムを分割して記述できるだけではなく、そのモジュール毎にファイルを管理運用可能なソフトウェア構造を提案している。新たなサービス･機能追加の際，既存サービスの再試験を不要とし、しかも既存の機能を引き継いで新サービスを、システムファイル更新をまたずに、随時、必要なオブジェクトのみを運用中システムに追加可能な環境を構築できる。

オブジェクト指向記述言語としては、汎用技術・プログラム資産・プログラマ・開発環境などを積極的に活用するために C++【21】の採用を前提とした通信ソフトウェア構成技術を提案している。

オブジェクト化されたプログラムを運用中の通信システムに、ユーザへ影響を与えることなく、投入可能な技術であるプラグイン機構について、新旧オブジェクトの同時走行技術と交換システム特有の共通データの引継ぎ技術を提案する【A3,A6】。これによりサービスの即応化を可能にする。プラグイン技術により従来（第一世代、二世代）行っている機械語によるパッチ投入ではなくプログラムソースレベルのパッチや，更に新規サービス（具体的には複数のオブジェクトを同時に）を運用中の通信システムに追加可能となる。

プラグイン動作の際、異常動作が生じた場合、プラグインした複数のオブジェクトを瞬時に元に戻す事が重要である。これを実現する技術であるプラグアウト機構について、旧オブジェクトへの復旧技術を提案する【A3,A6】。異状が発生した場合、プラグインしたオブジェクトを使用している呼だけを初期設定（１コール初期設定）する機能も同時に提供することにより高信頼通信ソフトウェアの構築が可能になる。

また、この様なプラグイン・アウト機構などをサポートするネットワーク管理について、分散システムへのＴＭＮ適用技術を提案する【A4,A5,A3】。これにより通信システム・通信ソフトウェアの効率的な運用管理が可能となる。

上記サービスを提供すると共に，今後多様化する通信システムのソフトウェア生産性向上を図るためにソフトウェアプラットフォームの構築について、カスタマイズ化技術を提案する【A2，A3，A5】。言語自体に並列性を持たせ，それを更に分散環境の実現にまで発展させることにより，プログラム設計者にシステムの作りを意識せずに済むシームレスな環境が実現できる。その技術についても本論文で提案する。

このプラットフォーム上でアプリケーションの実現を図る設計者は，対象とする問題，この場合提供するサービスのみに着目すればよくサービス設計の容易化が図れる。この様な環境下でのサービスプログラム自動生成に向けた実行可能な宣言的仕様化技術を提案する【D1】。

最後に，これらソフトウェアを開発するソフトウェア開発支援技術についても言及し，本論文で提案する各種技術の総合分析と評価を述べる。

このような一連の機能を実現するソフトウェアを本論文では、NOSES：Non-stop Service Enhancible Software と呼ぶ。以上を第一，二世代と比較してまとめると次の様になる。

(1)高･中･低の３位レベルのスケジューリングによる実時間処理
NOSES：リアルタイム性実現のために必須。但し、階層化により最下位の実行制御階層で実現。上位からサービス階層、リソース階層、実行制御階層の順に階層化し、スケジューリング機構は、実行制御で実現。また、サービスを実現するサービス階層には、　低レベルのプログラムのみで基本的には構成可能な方式。実行制御層とリソース階層を通信システムの共通プラットフォームとして提案。
(2)特殊トランザクション技術
NOSES：従来方式を廃止。階層化された最下位の実行制御層でタスキングを実現。
(3)冗長構成による障害自動再立ち上げ技術と救済技術
NOSES：従来技術を発展。プラグイン・アウト技術により更に信頼性とサービス性の向上を実現
(4)稼働中増設を可能とする為のトリー型データ構造
NOSES：従来技術を発展。ネットワークオペレーションシステム（OpS）との独立な発展を可能とする機能分担の確立。これにより多様なプロセッサ適用を可能にする。
(5)局データ・加入者データ管理
NOSES：従来方式を発展。プラグイン・アウト機構を可能にするために技術的なサポート機構を実現。

1.4 論文の構成と各章の概要

本論文では，本章で述べた高信頼通信ソフトウェア構成に関する研究成果をまとめたものである。本研究の全体構成を以下に示す。

本章である第 1 章で近年の通信ソフトウェアへの要求条件を論述した。そしてIT 分野などで提案されている先端技術でもこれらの要望に応えることが出来ないことを明らかにした。それに応えるための技術提案が本研究の狙いであることも述べた。そして 1.2 項要求条件と課題で，それらの要求条件に応えるためには 5 つの課題を解決する必要があることを示した。本論文は，これらの課題に対応した技術毎に章を構成している。第 2 章では通信システムのモデル化と階層化技術について論述すると共にオブジェクト指向通信ソフトウェア構成技術を明らかにする。これによりオブジェクト単位のサービス・機能追加の着脱が容易になる。第 3 章では，そのオブジェクト単位に着脱を可能にするプラグイン・アウト技術を明らかにする。リアルタイム性と高信頼性を踏まえた上でオブジェクトの追加·変更が可能になることにより，従来のシステムファイルによる立ち上げをまたずにサービス・機能の追加ができることから，今後の通信システム構築における必須機能である。第 4 章では，この様なプラグイン・アウト機能を多種多様な通信システムに適用するための共通プラットフォーム化技術を明らかにする。更に，システムの分散化をサービスプログラム設計者に意識させない言語レベルの分散 OS 技術についても提案する。第 5 章では，上記機能を実現する多様な通信システムとその通信システムからなるネットワークの管理技術を明らかにする。これによりシステムおよびネットワーク管理者による容易なプラグイン・アウトなどのファイル管理が実現可能になる。第 6 章は，本研究で提案する通信ソフトウェアの開発に向けた開発支援技術を明らかにする。階層化された通信ソフトウェア構成の中で通信サービスの形式的な仕様記述である宣言的実行可能な仕様記述を提案している。最後の第 7 章では本研究で提案する高信頼通信ソフトウェア構築技術を総括する。

第2章 通信ソフトウェア構成技術

【関連学術論文【A2】【A3】】

2.1 緒言

本章では，通信システムのモデル化，階層化技術及び設計法について述べると共にオブジェクト指向通信ソフトウェア構成技術を提案する。これによりオブジェクト単位のサービス・機能の着脱が容易になることを示す。また，サービスプログラムの設計が物理的なシステム構成に依存せず論理的なリソースのみを意識して設計できる事からプログラムの生産性向上が図れる。本章で提案するオブジェクト指向通信ソフトウェア構成技術によるプログラム生産性評価についても言及する。

2.2 モデル化と階層化手法

1) 大規模通信ｿﾌﾄｳｪｱへのｵﾌﾞｼﾞｪｸﾄ指向適用の狙い

ソフトウェアの複雑性については，各種論文【21】【22】で議論されている。その対策の１つとして，大規模ソフトウェアを単機能のプログラムとそれが扱うデータの組み合わせから構成しようとする試みがある。大規模なソフトウェアを見通しのきく大きさに分割する手法である。

大規模なソフトウェアを設計する上で注意することは，単機能のプログラム及びデータを体系化せず（階層化しないで）に扱うと，構成部品（単機能プログラム及びそれが扱うデータ）の数が多くなり，それぞれの構成部品間の相互関係が複雑になる。部品相互の依存関係を明確にせずに部品化を進めても，それらを組み合わせて大規模ソフトウェアを構築することが困難になる。また，構築できたとしても，維持管理費が高くなる。その様な作業をこなせる技術者は高度な技術が要求される。

これら構成部品間の相互関係（相互依存性）を見通し良くするために何をしたらいいかが重要なポイントになる。ただ単にインタフェースの種別を減らしたり相互に呼び合う回数を減らしたのでは意味がない。夫々のあるまとまった機能を実現する部品が，何のための部品であるか不明確になる。従って，構成部品間の依存性を体系化し見通しよくする事がポイントになる。

これを解決する手段として大規模通信ソフトウェアにオブジェクト指向を設計から製造・試験まで適用する手法を提案することが，本論文の１つの狙いである。

2) オブジェクト指向基本概念

ここで本論文が扱うオブジェクト指向を定義する。

第１章で述べた第二世代交換システムで試みられた機能ブロックによる大規模ソフトウェアの分割による設計・製造・試験が，複雑なソフトウェアの見通しを良くするために効果があった。その主な理由に次の２つがある。

１つは，トップダウンにより大きな“物”から順々に分割したことである。通信ネットワークは，端末，交換システムそして伝送装置に分割してモデル化できる。交換システムは，加入者系，通話路系，中継系及び各種トランク装置などに分割できる。或いは，交換システムをハードウェアとソフトウェアに分割し，ソフトウェアを更に加入者系サブシステム，中継系サブシステムなどの各種サブシステムに分割する方法が提案されている【15】。各サブシステムの中をさらに各機能ブロックに分割している。この機能ブロックを CHILL のモジュールに対応させてソフトウェアを構築している。これはオブジェクト指向モデルの“part of 階層”【21】に相当する。

２つ目は，共通部品化である。共通部品の品揃えをし，この機能ブロックはこれとこれの共通部品を組み込み，その機能ブロックの特有機能を追加して構築する，という捉え方である。これはオブジェクト指向の“kind of 階層”【21】に相当する。

オブジェクト指向では，この“part of 階層”をオブジェクトにより，そして“kind of 階層”をクラスにより実現している。従来進めてきたソフトウェア分割と部品化を言語レベルでサポートし，発展させた方式である（**図 3**）。

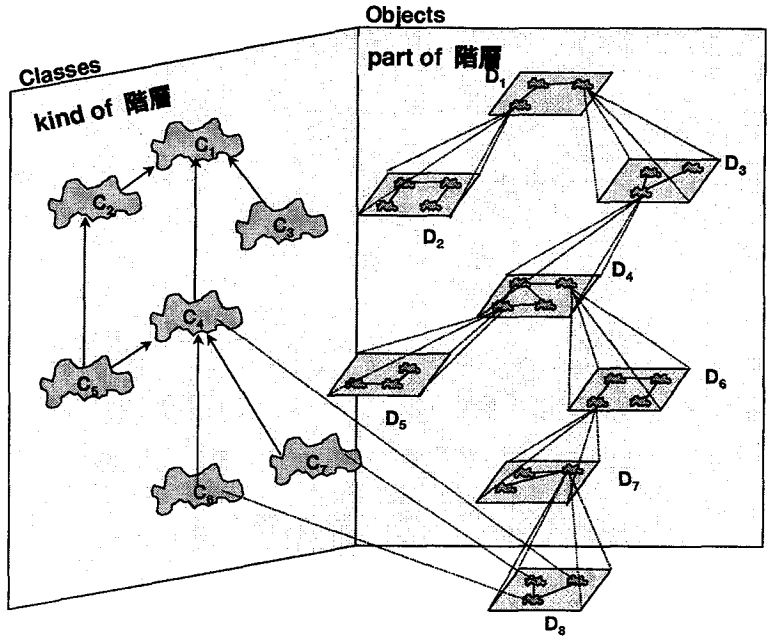


図 3　複雑なｼｽﾃﾑをモデル化する階層化手法

3) 通信システムのモデル化･階層化技術

オブジェクトモデル(part of 階層)を構築することにより次のメリットがある。

- C++の様なオブジェクト指向言語が持つ機能をフルに活用可能
- 設計段階でクラスの階層化を活用可能
- ソフトウェアの再利用ばかりではなく設計全体の再利用を促進
- システム全体の体系化による機能追加の促進
- 人の自然な思考法に従うことによる技術の引継ぎの支援

従って，サービス・機能追加変更時に対象となるオブジェクトが明確となり，次章で提案するプラグイン・アウトとの相乗効果が大きい。

交換システムを対象としたオブジェクトモデル構築法を以下に示す。

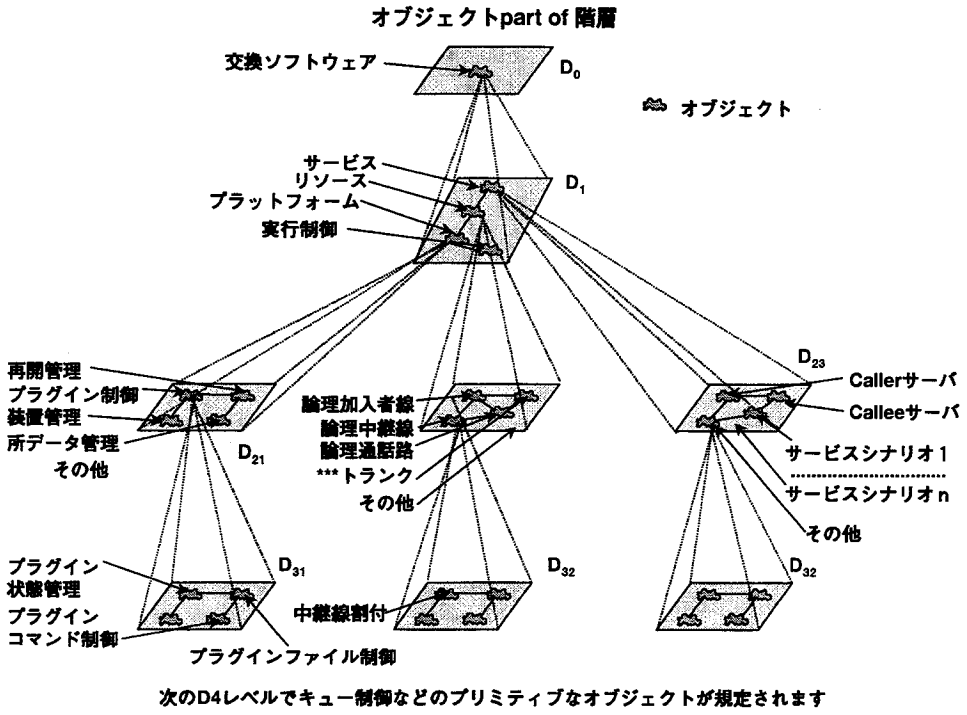


図 4　交換ソフトウェアのオブジェクトモデル

・交換システム全体を１つのオブジェクトで構成しているモデルを考える。このモデルでメッセージは、加入者線との送信・受信、中継線との送信・受信そしてオペレーションセンタとの送信・受信されるものとして定義出来る。このオブジェクトモデルは、端末の設計や相手交換システムあるいは、オペレーションセンタ（OpS）の設計などに必要なモデルとなる。

・交換システムを構成するハードウェアをオブジェクト指向モデルで捉えた場合を考える。この場合、加入者線・中継線・各種トランク等がオブジェクトとなる。このオブジェクトモデルは、ハードウェアの設計者にとって大切なモデルである。

・交換システムを構成するソフトウェアに着目して大きくサービス階層、リソース階層及び実行制御階層に分類する。この方式を採用すると各ソフトウェア階層に対応したプログラマのグループ化がしやすくなる。階層毎に必要な技術スキルが異なるので、技術者の育成などの面でメリットがある。これが効率的な維持管理につながっていくと期待できる。サービス階層を更にインテリジェントサービスを提供する高機能サービス階層と呼制御階層に分類した方がユーザによるカスタマイズ化などを考えるとベターである。また、色々なシステムで共通に使えるプラットフォームの提供を考えプラットフォームは、各システムで共通に使えるだけではなく、NOSES(non-stop service enhancible software)環境を提供する。次に各ソフトウェア階層毎のモデル化を行う。モデル化に当たっては、呼処理モデル、保守運用モデルそしてシステム制御モデルに分ける事が複雑なシステムを捉える上で都合がいいと考える。特に通信システムは高い信頼性を実現するためにかなり難しいシステム制御を実現しているので、それを切り離して検討することによって交換サービスの本質を見逃さずにすむ。

・サービス階層のオブジェクトモデルを構築するにあたって、先ず、呼処理モデルでポイントとなる呼に着目する。呼毎に呼サーバを設定し，呼毎に色々なサービスを各種の状況下（ダイナミックにサービスが変わって行くなど）で提供する方式を考える。そのためにサービス毎に異なるサービスシナリオオブジェクトを設定する。そうすれば、呼サーバは、状況に応じて色々なサービスシナリオオブジェクトに制御を委譲しながらユーザにサービスを提供する方式が実現できる。ユーザが、発信してから切断するまで呼サーバをユーザに対応して設定しておく。そして、ユーザの要望に応じたサービスシナリオオブジェクトを捕捉し、呼サーバがそのサービスシナリオオブジェクトに制御を委譲しながらサービスを提供する方式を考える。途中で、ユーザのサービス要望が変われば、それに対応したサービスシナリオオブジェクトを捕捉して、呼サーバは、新しいサービスシナリオオブジェクトに委譲する。更に、交換サービスの特性を考えると、発信系のサービスと着信系のサービスでは、特性が異なる。また、システムのビルディングブロック化や分散化に柔軟に対応するためにも呼サーバとサービスシナリオオブジェクトを発信系と着信系で分離する Caller-Callee モデルを採用する事にする。これで呼の状態管理が容易になると考える。
・次に、サービス階層の中で必要な、各種分析・翻訳など解析サービスを司るオブジェクトを設定する。
・リソース階層のオブジェクトモデルの構築にあたって、先ず考える事は、サービス階層にどのようにインタフェースを見せるか、つまりどのようなオブジェクトをサービス階層に対してビジブルにするかである。ここでは、サービス階層のプログラム設計を可能な限り楽にするために、交換機通話路論理装置モデルをサービス階層に提供する方式を採用する。論理加入線オブジェクト、論理通話路オブジェクト、論理中継線オブジェクトなど限られたオブジェクトにより仮想論理装置モデルを構築する。その配下に、個々の装置対応のオブジェクトとプラットフォームを構築する。
・プラットフォームのオブジェクトモデル構築にあたっては、プラットフォームの実現の目的から呼処理モデルよりもむしろ、システム制御モデルや保守運用モデルに関わるオブジェクトが中心になる。即ち、システム制御モデル面では、再開管理オブジェクト、プラグインオブジェクト、所データ管理オブジェクト、装置管理オブジェクトなどである。保守運用モデル面では、ＴＭＮ制御オブジェクト【23】【24】、MO(managedObject)翻訳オブジェクト【25】、(保守) コマンドメッセージオブジェクトなどがある。
・交換ソフトウェアを構成するプリミティブな機能をオブジェクトとして捉えることも大切である。それらには例えば待ちキュー制御等があげられる。これが最小のオブジェクトになる。
・それぞれの階層で、呼処理オブジェクトモデル、保守運用オブジェクトモデルそしてシステム制御モデルが構築されて全体のオブジェクトモデルが出来上がる。

以上の検討で出来上がった交換ソフトウェアのオブジェクトモデルが**図 4**である。オブジェクトモデルを三次元で書いていたのではスペース効率が悪いので二次元で表現すると、あるレベルから見たオブジェクト図となる。**図 4**に対応させると、**図 5**の外枠が D0 レベルのオブジェクト、階層分けが D1 レベルのオブジェクト、各階層内

のオブジェクトがD21~D23レベルのオブジェクトに対応する。D3レベル以下のオブジェクトは、**図 5**では省略されている。更に、細かなオブジェクトが存在することは見逃がされがちだが、大雑把な全体把握には**図 5**が便利である。また、階層間のオブジェクトの関係が表現しやすくなる。例えば、サービス階層からは、リソース階層の論理加入者線、論理通話路、論理中継線、論理音源、論理サービストランクオブジェクトしか見えないという事が表現できる。

この様にリソース階層が交換システムの論理リソースモデルをサービス階層に提供することによりサービス階層の設計者は論理リソースのみを意識すればよく，サービスプログラムの形式仕様記述が可能となる。この仕様化法と仕様からのプログラム自動生成技術については第6章の通信ソフトウェア開発支援技術で紹介する。

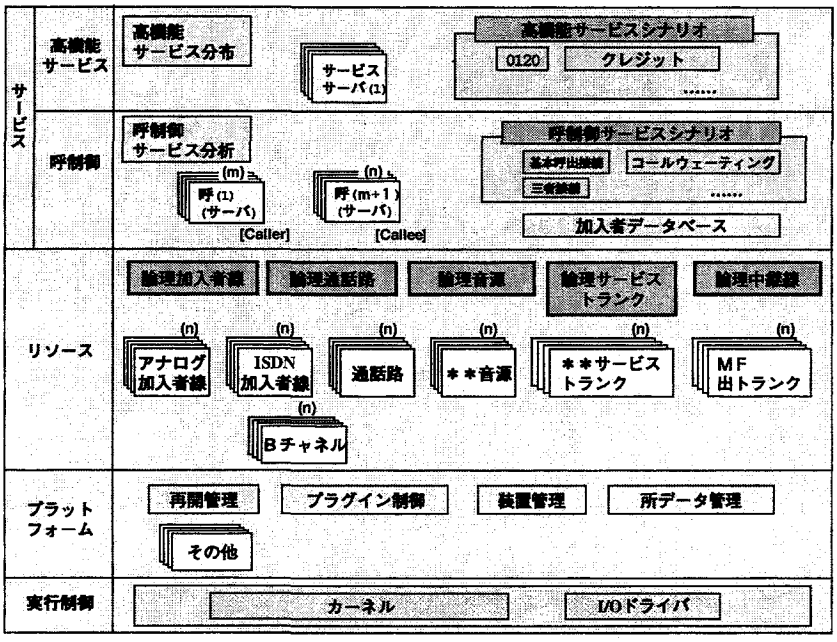


図 5　交換ソフトウェアの階層化

また，各階層が提供するインタフェースをどの様な形態で上位に見せるかを示したのが**図 6**である。呼処理などの様に厳しいリアルタイム性を必要とするインタフェースはAPI(Application Interface)本体としてサービス階層で使われる。つまりリソース階層が提供するシステムコントロールを直接サービス階層のプログラムが使うことになる。一方，サービス階層の呼制御が高機能サービスに提供するインタフェースはORB【26】やJava Applet【1】を経由して提供することも考えられる。

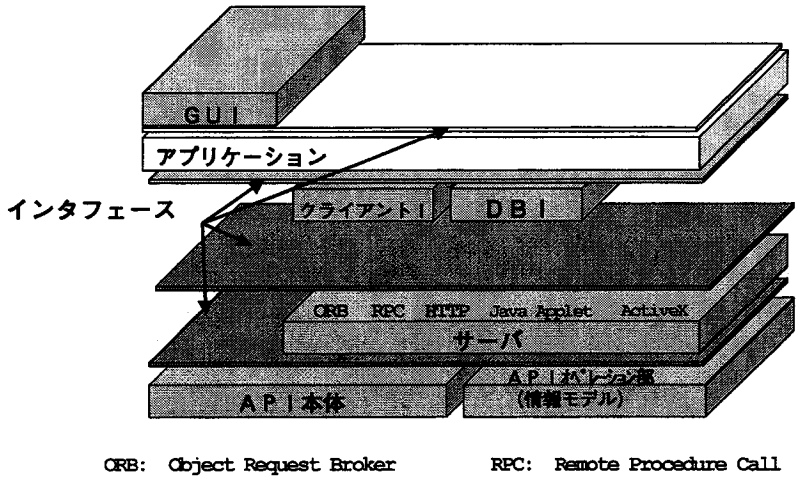


図 6 各インタフェースの参照モデル

2.3 通信ソフトウェア設計法

1) 基本的な考え方

オブジェクト指向ソフトウェアの開発は基本的に OMT(Object Modeling Technique)【28】に基づき行う。オブジェクト指向設計の流れを**図 7**に示す。また、モジュール化促進の観点から従来のソフトウェア設計法との違いを**図 8**に示す。

第一世代の交換ソフトウェアは、アセンブラで記述されているので Fortran や C 言語による手続き型というよりも、本図で示すとさらに左に位置するが、ここでは手続きとデータが渾然一体となっている、という観点から同一の分類としている。それを改善するためにモジュール化機能を持った言語 CHILL などの高級言語を第二世代の交換システムに適用する技術が検討された。

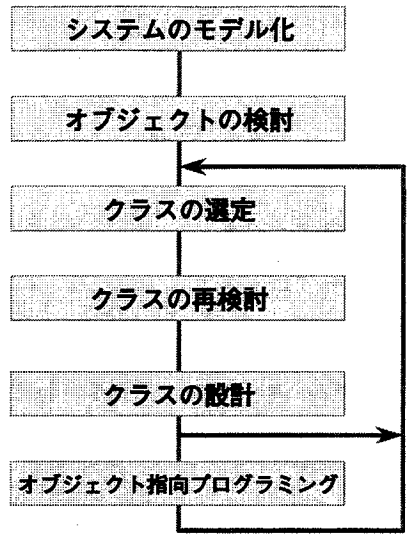


図 7　オブジェクト指向設計の流れ

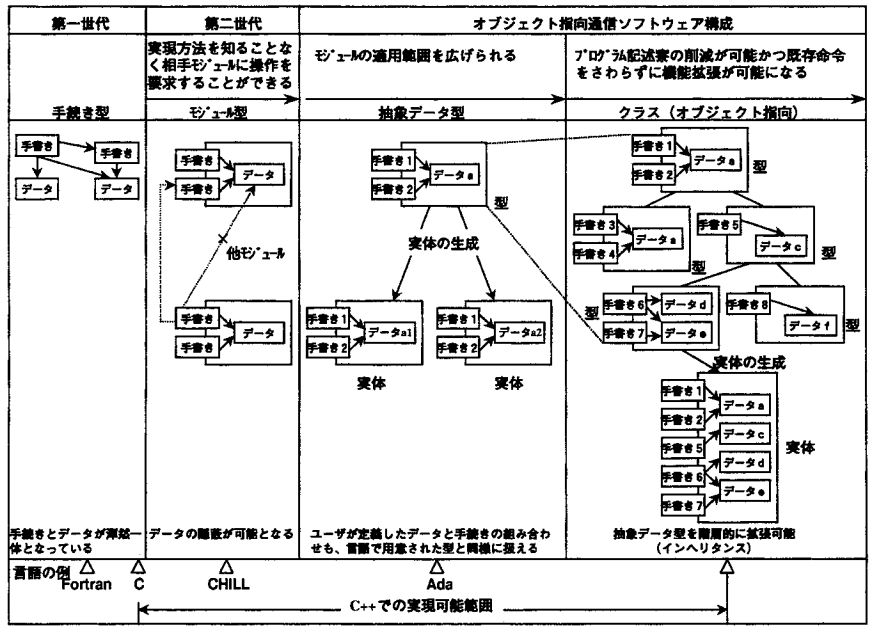


図 8 モジュール化の促進

CHILL が提供するモジュール概念で重要な事は，データの隠蔽が可能になることにある。大規模なプログラムを大勢で作り上げて行くとき、ある機能単位にシステムを分割し、その単位に設計分担を決めて仕事を進める事が効率的である。SE グループでシステム全体をどのように分割して設計するかを決めた後は、その機能単位に閉じて設計出来た方が、いつまでもシステム全体を見ないと各機能単位の設計やコーディングも出来ない場合に比べて大規模システムを効率良く構築できる。そのような設計手法をサポートする機能を言語が提供している。

モジュール概念を持たない C 言語でも大規模プログラムを開発できる。言い換えると C 言語でも分割設計が可能である。それは、C 言語が提供しているファイル化単位（コンパイル単位に相当）を活用して分割設計を行う方法である。そのファイル化単位を機能の分割単位に対応させあたかもモジュール化単位の分割に擬似する方式がとれる。ファイル単位とモジュール化単位とは、技術的には全く異なる目的を持った機能である。

例えば、後述する C++のクラスは、データ隠蔽の観点から見たモジュール化を促進したものでクラス単位でファイル化もできれば、その中のメソッド（関数に対応）毎にもファイル化することも可能である。クラスの中を細かく分けてファイル化してもデータ隠蔽の観点からは、クラスがモジュール化の単位である。

従って、**図 8**で示す第一世代の手続き型の言語は、そのユーザが意識／注意して使用すれば、その１つ上のモジュール化相当の機能を実現できるという事である。また、次の抽象データ型【18】も、言語がその機能をサポートしていなくても、その言語を使うユーザが注意して設計すれば CHILL などのモジュール型の言語で抽象データ型風の書き方は可能である。同じように、Ada や CHILL などを使ってオブジェクト指向風の設計を行なう事も出来る。しかし、これらの方法は、どれも設計者やプログラマに、より高度な技術を要求することになる。また、長い間の維持管理を考えるとその作法をいつまで守っていけるかが問題になる。

言語自体が、モジュール化、抽象データ型やオブジェクト指向機能を提供することにより、設計者やプログラマに余計な知識を要求せずとも一貫した作法でモジュール化プログラムや抽象データ型プログラムさらにはオブジェクト指向プログラムを書けるようになれば，より大規模なソフトウェア開発や維持管理の効率化や経済化が実現できる。

モジュール化機能により言語レベルでのデータの隠蔽が実現される。しかし、実際にプログラムの設計を進めていくと、同じ様なタイプ（型）が必要になる場合がある。その度にタイプをモジュールで定義していたのでは、効率が悪くなる。小規模プログラムの設計では，その様な機能が持つメリットは小さい。大規模プログラムを大勢で設計を進める様な場合、しかも、より低いコストで大規模プログラムを作り上げる場合，その効果が大きくなる。

一度，あるタイプをユーザが定義すれば，言語が提供するタイプと全く同じ様に何度でも使える機能，即ち，幾つかのタイプをユーザが登録（部品化）すれば、それを誰でも使えて、言語が提供しているタイプのように使える環境があれば、ユーザが自分で言語の機能拡張ができる事になる。この様なサービスをユーザに提供しているのが、抽象データ型言語である。

抽象データ型言語は、言語が用意している型（タイプ)、例えば、整数型の様に、あるデータを整数型で宣言するだけでそのデータが整数になるように、ユーザが定義した型を利用できる。しかも、抽象データ型は、データとそれを制御する手続きの組み合わせでユーザが定義できる方式である。これは大変便利な環境を言語が提供してくれたことになる。

設計者・プログラマ・試験者が上述で示す言語機能を大規模通信システムに適用することによりサービス追加の即応性と高信頼性を両立させることが可能となる。設計結果を**図 9**に、また、従来との比較を**表 1**に示す。

また、オブジェクト指向開発において導入する部品化の手法を**図 10**に示す。クラスを部品の基本単位とし、一部のライブラリ（流用した C 言語で書かれたライブラリや共通型定義など）を除いた全ての部品はクラスとして実現される。この部品化手法においてクラス間の継承技術を活用する事によりサービス階層のプログラム設計をより抽象化することが可能となり、呼処理などのサービスプログラムへの形式的仕様化技術の適用を図る事が可能になる。この事については本章で後述する。また、プラグイン及びアウトを実現する単位の明確化を図る事が出来る。

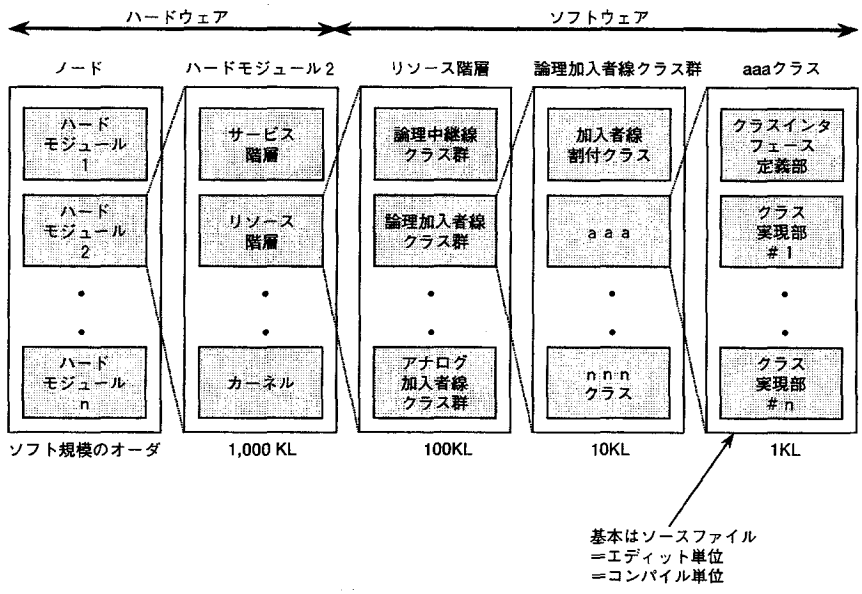


図 9　階層的構造

表 1　プログラム構造の比較

オブジェクト指向通信ソフトウェア		従来システム		規模のオーダ
階層	サービス階層 リソース階層 実行制御階層	サブシステム	加入者線サブシステムなど	100 KL
		機能ブロック	CHILLモジュール =コンパイル単位	10 KL
クラス群	論理加入者線など			
クラス	加入者線割付クラスなど			1 KL
(メンバ関数)	C++ソースファイル =エディット単位	ユニット等	CHILLプロシージャ =エディット単位	100 L
(部品)	クラスライブラリ バイナリライブラリ，・・・	(部品)	A部品，B部品，・・・	

Note：A部品、B部品、・・・
規模により分類

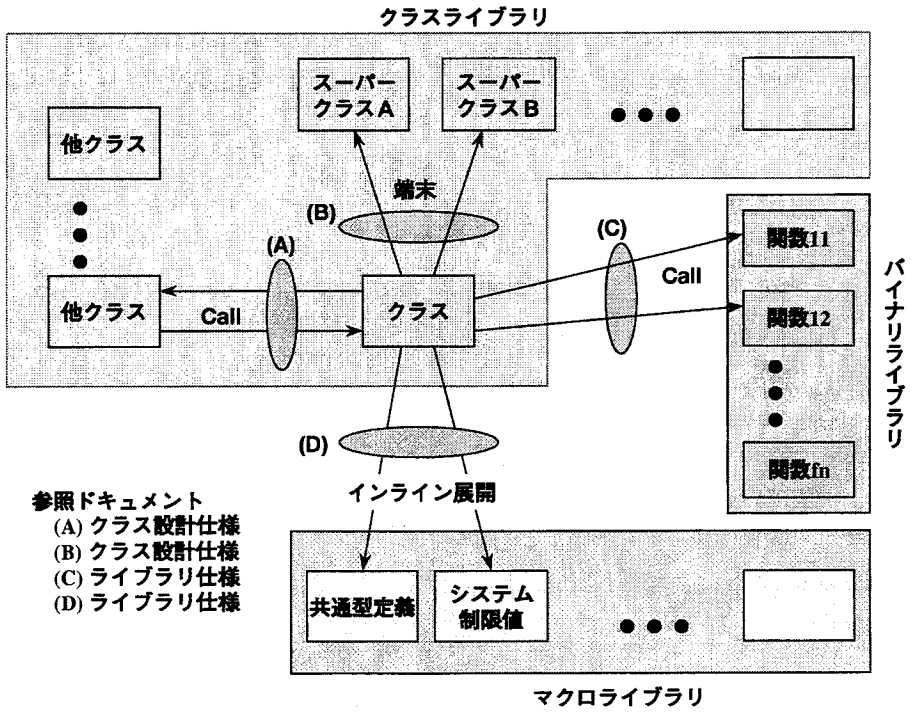


図 10　部品化の方法

2) 高信頼通信ソフトウェア設計法

交換システムソフトウェアのような大規模な通信ソフトウェアを構築するために必要となる基本的な設計法を提案する。また、具体的な例としては C++言語を用いて説明する。ソフトウェアの開発は、多くの技術者が協力して進める。従って、そこに適用されるソフトウェア技術ばかりではなく、目標を達成するために技術者をどのようにまとめ方向付けていくかも大切な要素である。前者については、ソフトウェアを構成する個々の部品である、クラスが提供するインタフェースのどれをパブリックに、そしてどれをプロテクテッドにするかを決める事が設計上の１つのポイントになる。後者については、どのようなソフトウェア開発体制を整えなければならないかが重要である。本研究では前者のソフトウェア設計技術に着目している。

システム仕様

システム仕様として大きく呼処理、保守運用そしてシステム制御に分けてモデル化する。

・呼処理モデル：通信サービスを実現する機能のモデル化。本論文では、発着分離モデルを採用する。Caller オブジェクトと Callee オブジェクトにサーバを分離し、これらサーバオブジェクトが、外部イベントにより決まるサービスに対応したサービスシナリオオブジェクトへ処理を委譲しながらサービスを実現する方式を採用する。
・保守運用モデル：保守運用サービスを実現するための“保守運用”機能のモデル化。試験オブジェクトにサーバ機能を持たせ、保守者からの外部イベントにより決まる保守サービスシナリオオブジェクトに処理を委譲しながら保守サービスを実現する方式を採用する。また、オペレーションシテム（OpS）とのインタフェースに TMN を適用する。それに対応した MO(Managed Object)モデルを考慮する。
・システム制御モデル：再開機能 、ファイル更新機能 、装置管理機能 、プラグイン機能に分けてそれぞれでオブジェクトモデルを設計する。

次にオブジェクトモデルを構築する。優れた設計をするための鍵は、“事実”を直接モデル化する事が重要である。尚、システム制御モデルについては第 3 及び 4 章に、保守運用モデルについては第 5 章で紹介する。本章では呼処理モデルを主に着目する。

呼処理モデル

簡単なサービス仕様を例に紹介する。

・例：『交換システムは、ユーザからの発信要求を受け付けると、ユーザの契約状況を調べた後、ダイヤルを受信する。受信した後、そのダイヤル数字を分析し、ユーザの要望が何であるかを把握する。そして、ユーザの要望に応えられる商品を持ってきて、ユーザに提供する。』

この仕様は、**図 5**が示すサービス階層で実現されるソフトウェアのサービス仕様を示している。加入線信号方式、加入者線交換システムの通話路構成などのシステムアーキテクチャに依存しない。どのような交換システムであろうと上記仕様は適用出来る。本仕様に従ったオブジェクトモデルを示す。

・サービス階層から見て、リソース階層にそのユーザ(1)に対応した論理加入線オブジェクト(SLO(1))を設定する。
・リソース階層の SLO(1)から発信要求がサービス階層に通知される。そのユーザ(1)にサービスを提供するユーザ対応サーバ(1)オブジェクトを設定する。発信系のサーバなのでこれを Caller(1)と呼ぶ。サービス階層にユーザ対応にサーバが用意されるモデル。
・ Caller(1)は、ユーザ(1)から、即ち SLO(1)から発信要求を受け付けたので、ユーザの契約状況を調べるためにデータベースオブジェクト(SubData(1))にアクセスする。
・SubData(1)が、ユーザの契約データを管理している。このオブジェクトが、ユーザの契約状況を判断して、その結果を Caller(1)に通知する。
・Caller(1)は、通知結果に基づき、論理加入者線オブジェクトである SLO(1)にダイヤル受信要求を出す。

・SLO(1)は、ダイヤル受信要求に従い、ユーザ(1)からのダイヤル受信制御を自律的に行う。勿論、サービス階層からは、SLO(1)しか見えないが、SLO(1)の配下で、色々な装置に対応したオブジェクトを制御している。
・SLO(1)でユーザ(1)からのダイヤルを受信すると、Caller(1)へ受信したダイヤル数字を通知する。
・Caller(1)は、SLO(1)から通知されたダイヤル数字をサービス分析オブジェクトへ送り、ユーザ(1)の要望を確認する。
・サービス分析オブジェクトは、ユーザ(1)の要望を分析する。必要であれば、自律的にデータベースオブジェクトなどにアクセスし、より広範囲のデータを収拾しながら分析する。その結果、ユーザ(1)が何を要望しているかが判明するので、それに対応した商品サービス(1)（サービスシナリオオブジェクト(1)）を選択する。これをCaller(1)へ通知する。
・Caller(1)は、サービス商品であるサービスシナリオオブジェクト(1)に従って、必要な各種オブジェクトへメッセージを送りながら、ユーザ(1)へサービスを提供する。

以上が、サービス仕様に素直に従って構築したオブジェクトモデルの 1 例である。**図 11**に呼処理オブジェクトモデル示す。

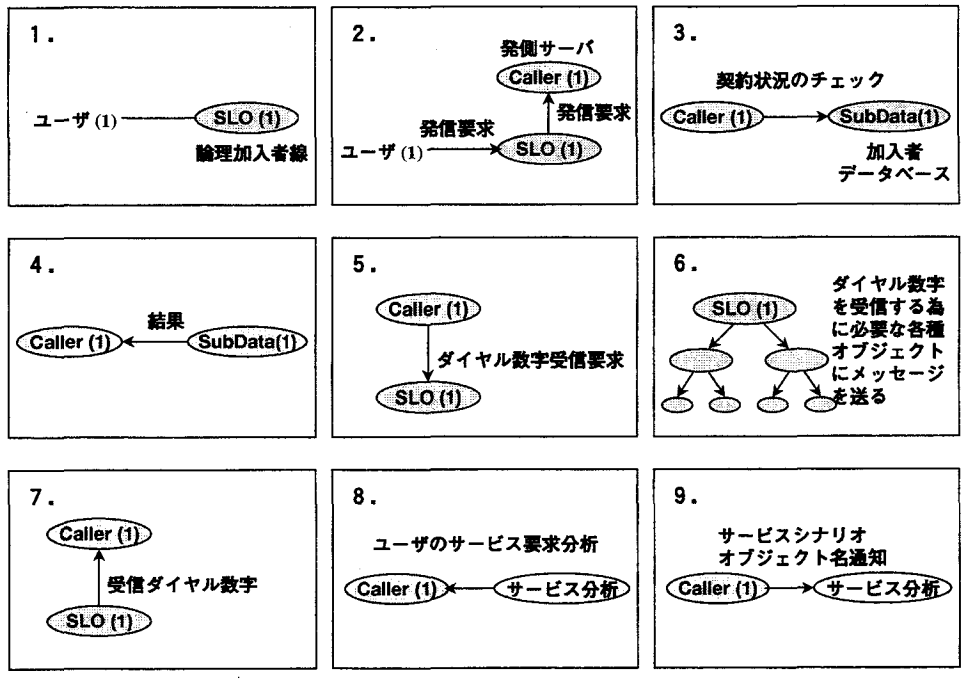


図 11 呼処理オブジェクトモデル

これにシステム制御モデルや保守運用モデルを加えて全体のオブジェクトモデルが構築される。その際、システム制御モデルに従った、新しいオブジェクトが呼処理モデルに加わると同時に、呼処理モデルで表現された各オブジェクトに必要なイン

タフェースが加えられる。例えば、初期設定を要求するインタフェース、呼救済を実現ために必要なインタフェースなどが考えられる。同じように保守運用モデルに従った、各種オブジェクトが追加されるとともに、呼処理モデルやシステム制御モデルで設定された各オブジェクトに必要なインタフェースが追加されていく。例えば、所データ設定インタフェース、保守情報通知インタフェースなどがありうる。このように順次モデルを拡張しながら何度も、前に戻りながらオブジェクトモデルをリファインしていく。

融通性、拡張性と再利用性

次に、融通性、拡張性と再利用性の考慮だが、次の様な戦略を採用した。

- ソフトウェアを階層化（サービス階層、リソース階層、プラットフォーム、実行制御階層）し、サービス追加時にサービス階層のみ考慮すればすむ環境を提供することによりサービス設計者は、可能な限りサービスのみ意識すれば良い方式を狙う。
- プラットフォームを構築し、各システムやハードモジュール間で共通に使用可能とする。
- プラットフォームをカスタマイズするツールを提供する事によりプラットフォームの再利用拡大を狙う。クラスの継承機能を活用する事により、既存機能に手を加えずに新たな機能を追加出来る環境を構築する。

以上を考慮して設計した交換ソフトウェアのオブジェクトモデルが、本章で説明した**図 4**及び**図 5**である。

標準クラス群の明確化

上述した全体像に従ってさらに詳細な設計に入る。開発を推進する上で、他からの再利用可能な標準クラス群がないか検討することは、効率的なソフトウェア開発を進めるために大変重要である。標準クラス群の候補に挙げられる製品としては、ファイル管理、データベース管理、TMN 制御などが考えらる。また、大規模なプロジェクトでは、各種のシステムを平行に開発する事も考えられる。各システム間の標準クラス群を決め効率的な開発分担を積極的に行うことも必要である。

本研究では、後者を推進するために各種クラス群からなるプラットフォームの構築を提案している。このプラットフォームを各種システム間で共通に使用しソフトウェア開発の効率化とソフトウェア技術者の共通文化を築く事が可能になる。これによりソフトウェアの再利用の促進に加えて技術者の円滑な流通を図れる。

交換システム分野では、交換システムが設置される局毎に異なる運用データ（所データなど）を運用中にユーザにご迷惑を掛けずに追加、変更をする必要がある。汎用製品では、この様な機能を提供していない。従って、通信システムへの再利用に当たっては、そのシステムに合ったカスタマイズ化を行う必要がある。もう１つの大きなカスタマイズ化は、NOSES を実現するために、プラグイン・アウト機構と初期設定機構の組み込みがある（次章参照）。

プラットフォームを各システム間で再利用し大規模ソフトウェア開発の効率化を図ることが重要である。また、プラットフォームの適用を拡大するためにはプラットフォームユーザの利便を考慮したカスタマイズ化ツールもプラットフォーム提供側が用意することが必要である。プラットフォーム上に各種アプリケーションを構築するためには、それ等アプリケーションのモデル、即ち呼処理モデル、保守運用モデル及びシステム制御モデルの統一とプラットフォームを含めた各種アプリケーションのソフトウェア設計作法の統一が大切である。つまりソフトウェア開発文化の統一である。これらが異なっていたのではシステム間で共通に使えるクラス群の定義ができない。また、ソフトウェア技術者の教育も効率が悪くなる。

オブジェクトに共通なデータや機能を抽出して、クラスを定義するが、最初の段階では、大雑把なクラス群の定義が必要である。段々と詳細化して行く。これらクラス間で共通な機能を提供しているもの同士を集めてクラス群とする。

オブジェクトモデルを使ってクラス群の選定を行なう。前述の呼処理モデルに着目した例を示す。

サービス階層：

- ユーザ毎にオブジェクト Caller(1)~Caller(n)がある。これは、発信系呼サーバクラスと着信系呼サーバクラスなどの幾つかのサーバクラスにより作られる呼サーバクラス群を設定する。
- データベースオブジェクト(1)も同様な考え方でデータベースクラス群を設定する。
- サービス分析オブジェクト(1)も同様考え方でサービス分析クラス群を設定出来る。
- サービスシナリオオブジェクト(1)もやはりサービスシナリオクラス群を設定する。
- など

リソース階層：

- サービス階層からみて、リソース階層がどのような論理通話路モデルを提供するか、どのようなモデルを提供すれば交換サービスを構築しやすい環境を提供できるか、を先ず決める。

・加入者線オブジェクト(1),,,(n)、中継線オブジェクト(1),,,(n)、通話路オブジェクト(1),,,(n)などがサービス階層に見える。それぞれに、加入者線クラス群、中継線クラス群、通話路クラス群を設定する。これらを論理リソースクラス群と呼ぶ。
・リソース階層内には、これら論理リソースクラス群の下に、サービス階層からは見えないが、共通線オブジェクト(1),,,(n)、各種トランクオブジェクト(1),,,(n)などがある。これらに対して、共通線クラス群などが設定出来る。
・など

次に、システム制御モデルに従い、リソース階層内に再開管理クラス群、プラグイン制御クラス群、装置管理クラス群など、また、保守運用モデルに従い、リソース階層内に所データ管理クラス群、マネージドオブジェクト翻訳クラス群、TMN クラス群などが追加される。

組立

本論文で提案するプラットフォームを各システムでカスタマイズ化し、各システムで必要なアプリケーションを搭載して組立が行なわれる。また、アプリケーションを構築する場合でも、NOSES で登録された各種クラス群を再利用可能である。勿論、“標準クラス群”といっても何にでも使えるというわけではない。ここで標準というのは、あくまでもここで提案している通信ソフトウェアの各種機能を実現する事を前提とした標準クラス群である。一部だけ持ってきても、そのままでは使える保証はない。

例えば、ある通信制御プログラムのクラス群だけを利用しようとしても修正が必要である。このプログラムは、運用データとして所データを持っている。所データを供給する環境が同時に提供されないかぎり、この部分の機能を利用する側の運用形態に合わした修正が必要である。提供されるクラス群をそのまま使いたければ、そのクラス群が前提としている設計作法をそのまま引く継ぐ事が必要である。

システムは、運用された後も当初考えていた範囲を越えて拡張されたり、再利用されたり、性能などをチューンされたりしながら絶えず変化していく。これに耐えうるソフトウェア設計を行う事が必要である。実際、初期設計から最初のリリースまでに、何度か要求条件が変更になることも現実に多々ある。このような条件に適合した“設計”というのは、融通性、拡張性そして再利用性を備えている必要がある。これを実現するためには、変更になりやすい、将来手が入りそうなところを可能な限り閉じ込めて限定出来る様にしておくことが最善な方法である。本論文では、階層化と

モジュール化（クラス化）によりこれを実現しようとしている。階層化によりサービスに関わるプログラムを上位のサービス階層に閉じ込め、かつサービス階層内でサービス対応にモジュール化（クラス化）することにより、サービス追加時の追加・変更箇所の局所化を狙っている。更に、クラス階層により追加サービスを既存サービスに手を加えずに実現可能な環境を提供しようとしている。具体的な技術を次に紹介する。

設計と言語との関係

材料の知識を持たずに建築設計は出来ない。それも強大な建造物を構築する程、材料や設備類などの深い知識が要求されるように、大規模ソフトウェアを構築するためには、そこで用いる言語や開発環境を熟知する必要がある。建築設計者が、その材料が持つ特性をいかに巧く活用するかを工夫する様に、ソフトウェア設計者は、インプリメントに用いる言語が提供する機能を最大限に工夫し、目的を達成する事が必要である。また、言語が提供している機能では不十分な場合、設計者やプログラマは、言語をサポートする各種ツール（リンカ、プリプロセッサ、ファイル化ツールなど)、オンラインソフトウェア上のカーネルなどのオペレーションシステム(OS)やアプリケーションの書き方による工夫などで補う必要がある。

言語の発展も、問題意識を持ったソフトウェア設計者やプログラマの要望に応えて発展してきた。前述したモジュール化の促進などがその一例である。手続き型の言語からモジュール化、そして抽象型に発展し、現在その集大成としてオブジェクト指向型言語が提案されてきた。オブジェクト指向型言語である C++は、従来の手続き型言語である C を拡張した C 言語としての機能、抽象型及びオブジェクト指向をサポートした言語である【20】。

C++は、大規模ソフトウェアを構築する上で従来に比べ大変有効な機能を提供している。それをどのように高信頼通信ソフトウェア構築に活用するかも本書の 1 つの狙いである。NOSES の様な本研究で狙いとする機能を実現するには、言語（コンパイラが提供する機能を含んで）が提供する環境のみでは不十分なのでツール， アプリケーションや OS での工夫が必要になる。どんな機能を言語にたより、それ以外をどのように実現するかを紹介する。

C++が提供する環境でポイントになる機能が、クラスの環境である。クラスは、設計者やプログラマが定義できるタイプである。あたかも言語自体が提供しているタイプと同じようにユーザが定義したタイプを使うことが出来る。しかも、機能継承機

能を使って、既にあるクラスを変更することなく、差分を用いた機能追加により全く新しいクラス（タイプに相当）を作り出すことが出来る。また、クラスは、“情報の隠蔽”のメカニズムを提供してくれる。その外にも

- バーチャル関数：これを使えば、関数間の呼び出しをダイナミックにしかも、スタティックなタイプチェックをくずさずに実現できる。バーチャル関数で実現されるダイナミック呼び出しは、間接ジャンプ方式により実現される。Smalltalk【29】の様なサーチ方式では、リアルタイムなシステムでは使えない。また、Smalltalk は、スタティックなタイプチェック機能も提供していない。このお陰で、継承機能とポリモフィズムが巧く言語レベルで実現されている。サブクラスで引き継ぐ関数を親クラス側でバーチャル関数指定すればすむ。サブクラス側で親と同じ関数名であるが違った機能を実現する場合、バーチャル指定により上書きが出来る。
- テンプレート機能：これによりパラメータ化したタイプを定義できる。例えば、ビット列を表すクラスを作成する場合、普通に考えると、ビット列の長さに応じたクラスをそれぞれ作るか、ビットの長さによらずに使えるクラスを作るかのどちらかになる。前者は、さまざまな長さのクラスを作成しなければならず、クラスの数がどんどん増えてしまう。後者は、どの長さのビット列でも同じプログラムで扱うため、ビット列の長さの違いに対する型チェックができなくなるか、実行時にチェックをする必要がある。従って、どちらの方式も一長一短がある。そこで、記述としては一つでありながら、長さの違いにより必要なクラスを作り出す機能として、テンプレート機能が用意されている。これは、クラスの型紙（クラスのクラス）であるクラステンプレートというものを一つだけ書いておき、必要な人がこのクラステンプレートから自分の必要に合わせたクラスを作り出せるという機能である。作り出されたクラスをテンプレートクラスと呼ぶ。この機能を用いると、もともとのコーディングは１つだけで、必要に応じたクラスを簡単に作り出すことができ、コンパイル時に型チェックができるということになる。従って、先ほど述べた両方式の良いとこ取りをした方式を提供していることになる。
- アクセス権制御：これを使えば、データのガードが可能になる。

アクセス権制御

言語が持つアクセス権制御を活用することにより呼処理プログラムの不正な処理によるシステムへの影響を最小限に押さえることができる（**図 12**）。

CHILL との比較で説明すると、クラスが CHILL モジュールに相当する（勿論、クラスには、データの抽象化とオブジェクト指向実現機能があるが CHILL モジュールにはない）。パブリック(public)が CHILL の GRANT 宣言に、プライベイト(private)が CHILL のユニット（UI）に相当する。プロテクテッド(protected)とフレンド(friend)が新規機能である。前者は、継承機能で親クラスが宣言するとサブクラスだけがアクセス可能になる。インタフェースで他のオブジェクトがアクセス出来るメンバ関数を

・クラスによりモジュール化が促進

・メンバ関数とデータには３種類の属性が設定可能

・必要ならばクラスの枠を越えたアクセスも可能

private, protected, publicで指定されるメンバ関数とデータへのアクセス

ガード 厳しい ↕ 緩い

	自クラス	サブクラス	フレンド	その他
private	○	×	○	×
protected	○	○	○	×
public	○	○	○	○

○：アクセス可、×：アクセス不可

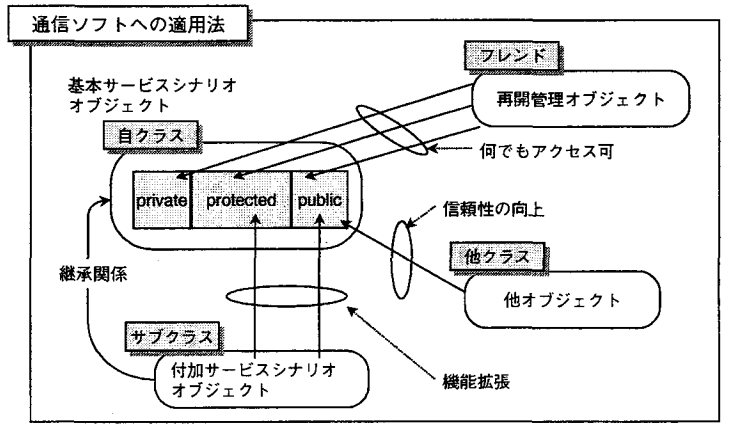


図 12 データのガード機構

パブリックに宣言し、オブジェクト内（クラス内）でのみアクセスするメンバ関数をプロテクテッドに宣言すればいい。プライベートは、呼情報（呼の状態など）に適用し、その情報を管理しているクラスだけがアクセス出来る。この時、再開管理側をフレンドにしておけば、この呼情報を見ながら呼救済することにより信頼性を上げながら且つ、効率よく実現できる。

クラスの継承を活用

新たにサービスを設計するにあたりゼロから設計をしたのではソフトウェア生産性の観点から不効率である。それまで培ってきた資産（ソフトウェア部品ばかりではなく知識を含めて）を活用できることが必要である。その手段をオブジェクト指向言語が与えてくれる（**図 13**, **図 14**）。

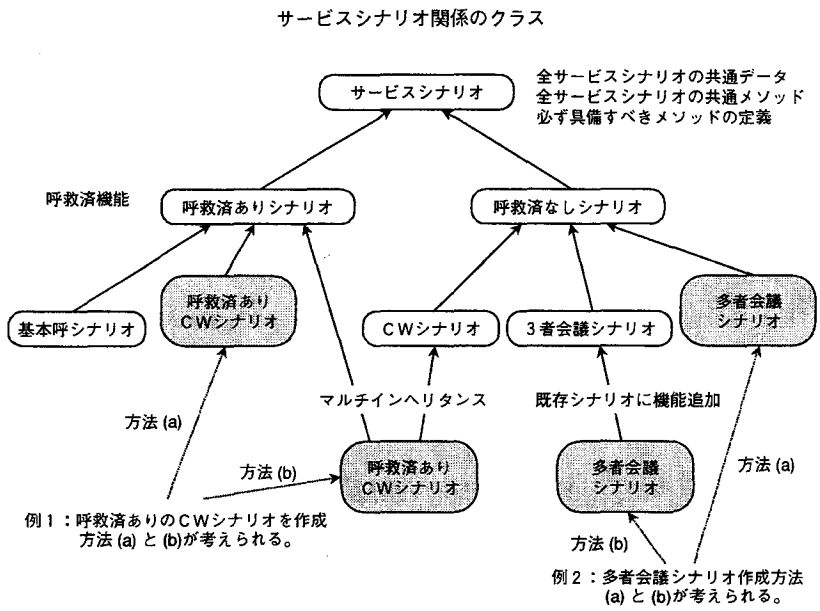


図 13 サービスシナリオクラスの継承関係

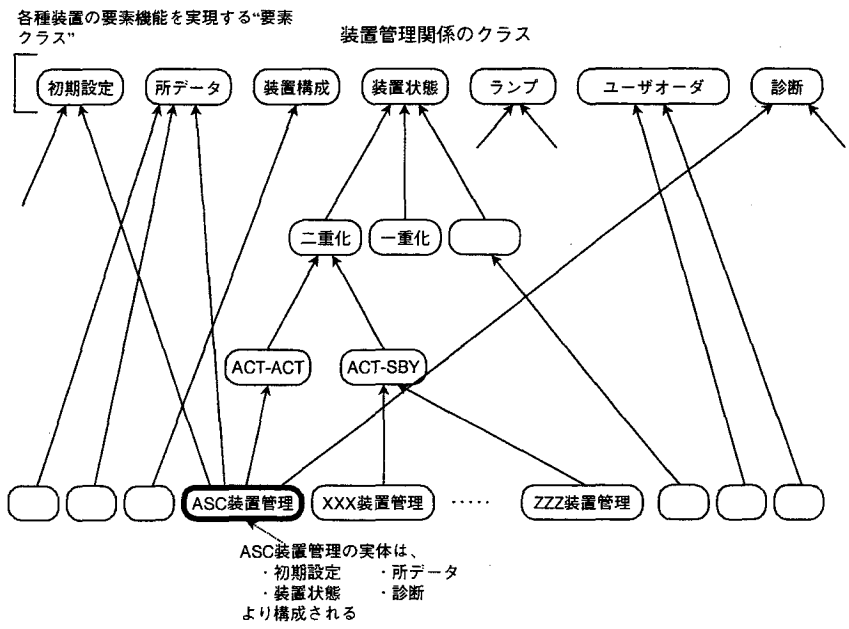


図 14 装置管理クラスの継承関係

2.4 評価

1) プラグイン・アウトへの適用性

第 1 章で述べた第二世代のシステム設計との比較の中でプラグイン・アウトへの適用性について述べる。各種トランク（中継回線など）の仕様設計書は、トランク種別毎に仕様設計する（**図 15**)。装置毎に仕様設計することは従来から実施されている。この方式は自然な設計法であると考える。MFOGT ，MFICT 毎（中継回線種別毎）に状態遷移図を記述する。しかし、この装置毎の仕様を CHILL などの言語を用いて記述する際、従来は多種類ある装置毎に CHILL モジュールを対応させていたのではメモリ量が増えるので、それらに関係なくトランク制御モジュール,トランク監視モジュール、 リンク制御モジュールなどの CHILL モジュールにまとめられて実現される（**図 16**)。仕様は、各装置毎に、プログラムは、それに対応せずに全部一括して記述されている。この例だと仕様書は、オブジェクト指向モデル風に書かれていたが、CHILL などの従来の言語を使って装置毎、即ち、オブジェクト毎にプログラムを記述していたのでは無理があり記述量が増加する。トランクの種別が異なっても共通機能は多い。そのためにトランク制御モジュール,トランク監視モジュール、 リンク制御モジュールで各トランク制御に必要な機能をまとめて実現することになる。そして、各 CHILL モジュールの中で共通ルーチンを駆使すればプログラム規模が小さくなる。この様な設計では最小の開発量で実現出来るが、あるトランク機能が変更になったにも関わらず全てのトランクを制御するプログラムに影響を与えることになる。また、トランクを制御するソフトウェアの設計技術者は、常に全てのトランクに関する知識を持つ必要がある。

これでは、あるトランク制御に関わるサービス・機能追加変更に伴い、その追加変更に関わらない全てのソフトウェアに影響を与えることになり、サービス・機能の追加変更を安全に実施することができない。

一方、本論文で提案するオブジェクト指向設計によりプラグイン・プラグアウトの実現が容易になる。

新しい機能は差分だけを記述したクラスを追加すればよく、既存ソフトウェアを変更せずにすむ可能性が高くなった。しかも、サービス仕様設計に対応したソフトウェア構造（**図 16**）が実現出来ることから次章で紹介するプラグイン・アウトの適用が安全に且つ確実に実現出来る。

本仕様に従ったソフトウェア構築を以下に示す。

・中継線トランクのユーザは、先ず中継線管理クラスへ必要とするトランク種別を指定し、中継線の捕捉を要求する。
・中継線管理クラスはユーザからの要求に従い、トランク種別から必要な中継回線に対応するトランクを捕捉する。このトランククラスが責任を持って該当する物理中継回線の制御を司る。従って、このクラスが責任を持つ物理中継回線の状態制御や制御機能をユーザの要求に従い該当するオブジェトが実行できるための仕様が記述される。
・中継線管理クラスは、生成されたトランクオブジェクト名をユーザに通知するために必要な仕様が記述される。
・ユーザは、通知されたオブジェクト名に対して必要な作業に対応したメッセージをやり取りしながら仕事を進めていく。

この様にサービスや機能の追加は、その仕様を記述したクラスのプラグイン・アウトに相当することになる。

オブジェクト指向設計は、本論文が狙いとするサービス・機能のプラグイン・アウトに適した方式である。

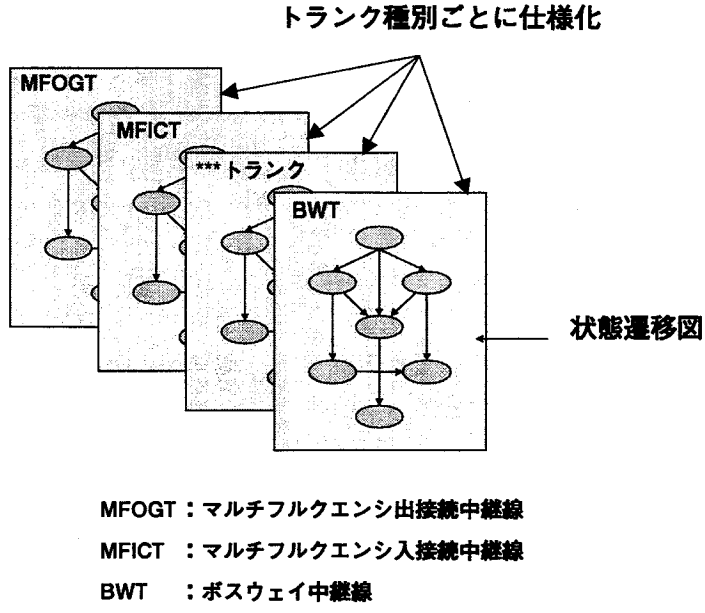


図 15　各種トランク装置の仕様

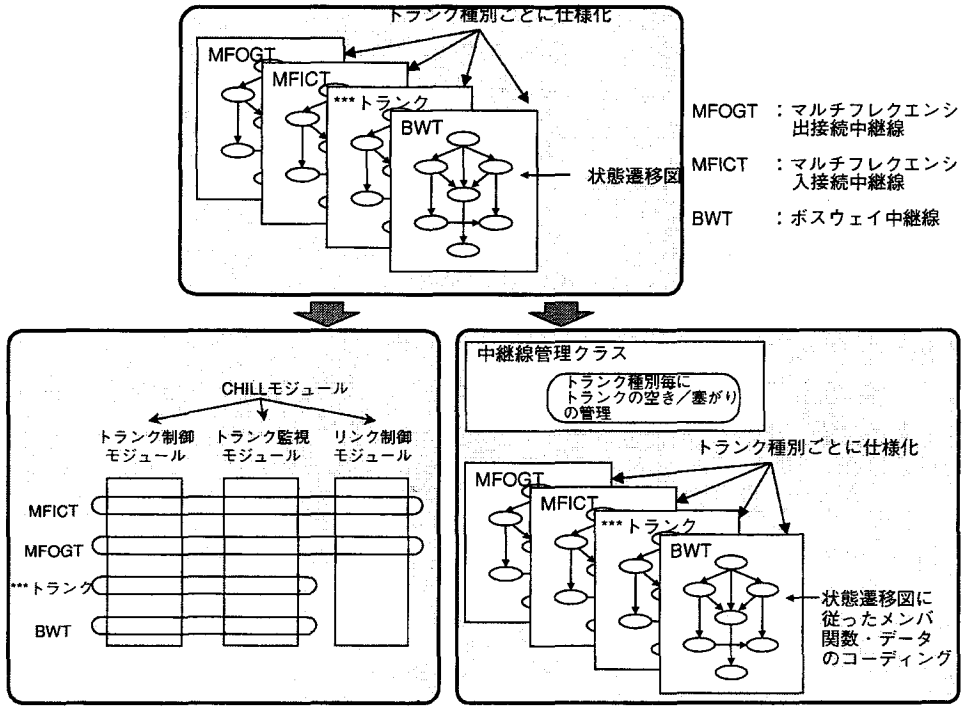


図 16　従来実現方式と本提案方式の比較(トランク装置に着目)

2)処理能力評価

C++特有の機能はほとんどがコンパイル時に解決されるため、同等の処理を記述したCプログラムと比べても処理ステップの増加はほとんどない（**図 17**）。

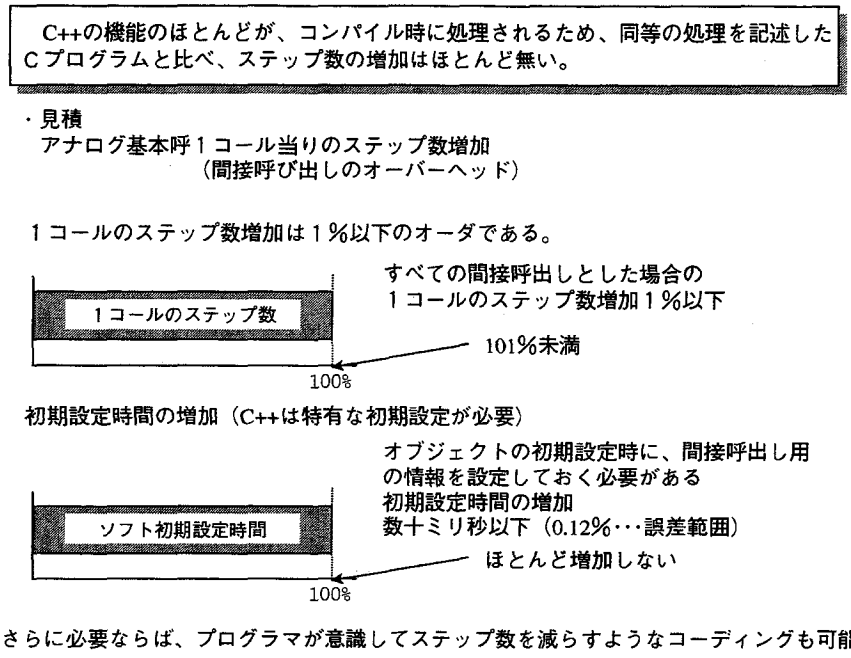


図 17　ダイナミック数増加要因の分析

2) 生産性評価

オブジェクト指向通信ソフトウェア構成技術により複雑なシステムの代表格である通信システムを把握可能な単位まで抽象化レベルを落とすことが可能となった。これによりサービスプログラムの設計が物理的なシステム構成に依存しない形で実現できる環境が提供可能となった。これによる生産性への効果が大きい。所内実験機による評価結果では，サービス階層プログラムは第二世代の交換システムと比較し 2 倍の生産性向上が，リソース階層及び実行制御階層を含めたトータルな生産性は 15%の向上を得ることができた。これらの評価は，NTT で機能追加・変更した過去 2 年間のサービスを全て本システムに追加・変更して評価したものである。

2.5 結言

通信システムのモデル化と階層化技術手法について述べると共にオブジェクト指向通信ソフトウェア構成技術を提案した。これにより複雑なシステムの代表格である通信システムを把握可能な単位まで抽象化レベルを落とすことが可能である。更に，

第 3 章で提案しているプラグイン・アウト技術への親和性も優れており，オブジェクト単位のサービス・機能追加の着脱が容易になることを明確にした。また，サービスプログラムの設計が物理的なシステム構成に依存しない形で実現できる環境を提供可能になるため，サービスプログラム設計に向けた形式的仕様記述の適用範囲が大幅に増大することを示した（第 6 章と関連）。

所内実験機による評価結果では，サービス階層プログラムは 2 倍の生産性向上が，リソース階層及び実行制御階層を含めたトータルな生産性は 15%の向上を得ることができた。これらの評価は，NTT で機能追加・変更した過去 2 年間のサービスを全て本システムに追加・変更して評価したものである。

更に，この様な効果を処理増で払った税金は処理時間で数%増，システム初期設定時間で数%の増加で実現できる。

また、次章で提案するプラグイン機構と連動することによるソフトウェア生産性の向上は大きく、この結果を踏まえ，本研究が提案する通信ソフトウェア構成技術が NTT のこれからの交換システムの全てに適用されることが決定され，1996 年 12 月に導入された PHS サービス向け大容量交換システムを皮切りに，1997 年導入予定の大容量 ISDN 交換システム及びこれから導入される ATM 交換システムに適用される。

付録

2.1 通信ソフトウェアの“複雑さ”

ソフトウェア開発で最も基本的な問題は、ソフトウェアが本質的に持っている“複雑さ”にある。この“複雑さ”に対処する基本的なやり方として、“取り扱いやすい大きさに分割する”ことである。この手法は、まさに第1章で述べた交換ソフトウェアの第二世代での主要テーマであった。そして、本研究の主要課題でもあり、更に、運用中のシステムに影響を与えることなくサービスの追加変更が可能なソフトウェア構成と設計法の実現に向けた技術の提案が本研究の目的である事はすでに述べた通りである。

この分割手法は、色々な方法が試みられている。特にモジュールあるいはクラスをソフトウェア設計に適用する場合は、プログラムをそのモジュールあるいはクラスの内部仕様とその外部インタフェースに分けて設計する方式がある。こうする事によって、設計フェーズを各モジュールの外部仕様を決める段階と内部仕様を決める段階に明確に分ける事が出来る。

ソフトウェアが複雑な理由として4つの要因を挙げることができる。(1)対象とする問題領域自身が持つ複雑性　(2)開発管理の複雑さ　(3)ソフトウェア自身がもつフレキシビリティ性そして(4)対処システムの動作を仕様化する複雑さ、である。

対象とする問題領域自身が持つ複雑姓

交換システムやロボットなどを制御するソフトウェアを開発するためには、それらシステムが持つ機能を明確に理解する事がまず難しい課題になる。さらに、これらシステムの開発にあたって、ソフトウェアの再利用性、コスト、寿命（その技術がいつまで持ちこたえられるかなど-----サバイバリティ)、信頼性などが要求される。即ち、システムのユーザ要望を考慮しながら開発する必要がある。

そして、システムのユーザと開発者間での不一致が、対象とする問題領域をますます複雑にする。システムのユーザが、望むニーズをシステム開発者が理解できる形式的な表現を用いて示すことは、今までのシステム開発の経験からみて相当難しい問題である。また、更に問題を難しくしている事は、ソフトウェア開発中にユーザの要望が変わることである。

また、大規模ソフトウェアは、それ自身が大きな資本投資である。新しい要望が示される度に既存のソフトウェアをスクラップに出来る余裕が、いつでもあるわけではない。新しい要望を取り込みながら発展していく必要がある。５年前、１０年前あるいは１５年前に設計したプログラムと一緒に動作しながら、新しいニーズに応えるようにソフトウェアを発展可能な構造にする必要がある。これは大変な労力がいる仕事である。ソフトウェア維持管理を難しくしている１つの大きな要因と考えられる。

開発管理の複雑さ

大規模なソフトウェア開発を行うためには、多くの技術者がチームを形成して進めていく。この技術者間でいかに設計思想の一貫性を保って進められるかが１つのポイントになる。技術者の人数が多いほど、また新しいチャレンジが多いほど難しくなる。即ち、最も信頼性が望まれる大規模システムほど開発管理が難しくなる。

ソフトウェア自身がもつフレキシビリティ性

建築業者は、決して自ら営林業を経営することはない。建築技術者と営林技術者がもつ技術の違いがあまりに大きく、両者を一緒にした企業・組織をおこし経営する事は、その非効率さから考えられない。しかし、ソフトウェア開発に於ては、事情が異なる。ソフトウェア開発者は、抽象的なクラスからできるブロックを組み立てることからそのブロックの材料である各種関数やデータの設計、必要であれば、その材料を産み出す“種”である言語のコンパイラなどのツールも自ら設計する必要がある。ソフトウェア開発は、労働集約型ビジネスである。

対処システムの動作を仕様化する複雑さ

無風状態の中で、ボールを投げた時、そのボールがどの様な放物線を辿って飛んでいくかを示すことは比較的簡単なことである。

大規模なシステムを構築する場合、そこには、何百、何千という変数を、たった１つの制御の流れの中でも処理する必要がある。例えば、通信システムでの１つの“呼”のある時点での状態を正確に知るには、それが関わる全ての変数の変化、現状態、現アドレスや呼出す側のプロセスのスタックの中身、これら全てを把握する必要がある。Parnas が以下の事を指摘している【30】。

- “連続な機能としてあるシステムを表現できると、そのシステムへの小さな機能追加は、小さな変更ですむ。しかし、コンピュータのようにディスクリートな状態（離散的な状態）を持つシステムは、

有限状態マシンとしてモデル化出来るが、小さな変更でも、この状態が爆発的に増えてしまう傾向がある。このような事が生じないように、システムの振る舞いをある単位に分割することが有効である。”

実際、離散的な状態をもつシステムの場合、そのシステムへの外部的な事象は、システムの内部状態に必ず影響を与える。これが、システムのデバッグを難しくする要因になっている。大規模システムのデバッグを全ての外部事象を考慮して完全に実施することは不可能である。数学的に立証されたツールもなければ、大規模システムの振る舞いを完全に表現可能な知的な能力も、まだない。ある許容範囲内での信頼性に満足せざるを得ない。

本研究では、2.1 項で述べたオブジェクト指向設計法に基づき“複雑な通信システム”を体系的に分割管理する。

第3章 プラグイン・アウト構築技術

【関連学術論文【A3】【A6】】

3.1 緒言

本章では，第 2 章で明確になったオブジェクトを運用中の通信システムに着脱可能とするプラグイン・アウト技術を提案する。これによりサービスを受けているユーザに影響なく運用中の通信システムにサービス・機能の追加･変更が可能になる。リアルタイム性と高信頼性を踏まえた上でオブジェクトの追加･変更ができることにより，従来のシステムファイルによるシステム立ち上げをまたずにサービス・機能の追加ができるため今後の通信システム構築における必須機能と考える。

3.2 課題の明確化

3.2.1 狙い

先ず、従来のプログラマが機械語で対処していた緊急パッチを高級言語レベルでバグを修正できる環境を提供する。第１章で述べた第二世代までの交換システムでは、全国にある通信システムを専用プロセッサなどで統一し、プログラムやデータのメモリ割付を全システムで同じにする方式を採用している。この様な環境では機械語のパッチを１本作り全国に配付するだけでよい。しかし、高度なサービスを提供する通信システムが何時までも全システムに同じ運用ファイルを搭載し、しかも同じアドレスに割り付けられている状況が続けられるとは考えられない。ユーザの要望によるカスタマイズ化が今後益々重要になってくる。例え同じ機能でもユーザの要望によっては異なるアドレスに割り付けられる事が予想される。更に、汎用プロセッサを前提としたシステムに従来の機械語パッチで対処すると、プログラマや保守者に、システムに適用されているプロセッサの種別と、プロセッサ毎に異なる機械語の意識が必要になる。汎用プロセッサを適用しプロセッサ技術の親展の恩恵を受けられるシステムを構築するという事は、システムに複数種類のプロセッサが使われる可能性が大変高くなる。この様な状況下で機械語によるパッチの維持管理を技術者に強いるのは避けるべきである。この様な問題に対処する為に、プロセッサに依存した機械語やそのシステムに依存したメモリ割付は、可能な限りそのシステムに閉じて実現し、外部に対して不要な情報を隠蔽する事が大切になる。プログラマや保守者が高級言語のみ意識すればよい緊急パッチを、本論文では基本プラグインと呼ぶ。

また、近年　ユーザのご要望に応えて、早期に新サービスを実現できる事が強く望まれている。新サービスの追加を運用中のファイルで、しかもサービス提供中のユーザに影響を与える事なく実現する必要がある。これを可能にするサービスを応用プラグインと本論文は呼んでいる。

運用中のファイルを追加・変更するには、プラグイン機構に高度な信頼性を保証する事が要求される。基本及び応用プラグイン実行時、異常が生じた場合は、その異常に遭遇した呼のみ初期設定する（１コール初期設定）ことによりシステム全体へ影響が派生しないなどの工夫が必要不可欠である。プラグインを失敗した新オブジェクト（変更後のオブジェクト）を速やかに排除し旧オブジェクト（変更前のオブジェクト）に戻す必要がある。この機構をプラグアウトと呼ぶ。

以上の様に、ユーザに影響する事なく運用中ファイルのバグ対処（基本プラグイン）や新サービスを提供する為に新オブジェクトの追加・変更（応用プラグイン）を実現すると共に、異常が発生した際にプラグインされた新オブジェクトを排除するプラグアウト機構やその異常の影響を最小限に押さえるためのソフトウェア初期設定技術の確立が重要であり、本論文はそれらの技術を提案するものである。また、プラグイン・アウトするためのファイルの転送や保守コマンドの転送機能などを実現して初めて、安心してプラットフォームをユーザが使う事が出来る。このような環境を提供する事を NOSES : Non-Stop Service Enhancible Software の環境を提供するプラットフォームと呼んでいる。本論文の１つの狙いがこの NOSES 環境を提供するプラットフォームの構築技術にある。

3.2.2 課題の捉えかた

基本・応用プラグインの実現モデルを第２章で紹介した通信ソフトウェアのオブジェクトモデルにより表現すると**図 18**が示す運用中ファイルへのオブジェクト追加・変更モデルとして捉えることができる。即ち、すでに運用されている既存オブジェクトの入れ替えと全く新規のオブジェクトを運用されているファイルに組み込むモデルとしてモデル化できる。基本・応用プラグインは、この２つのモデルの組み合わせにより実現される。

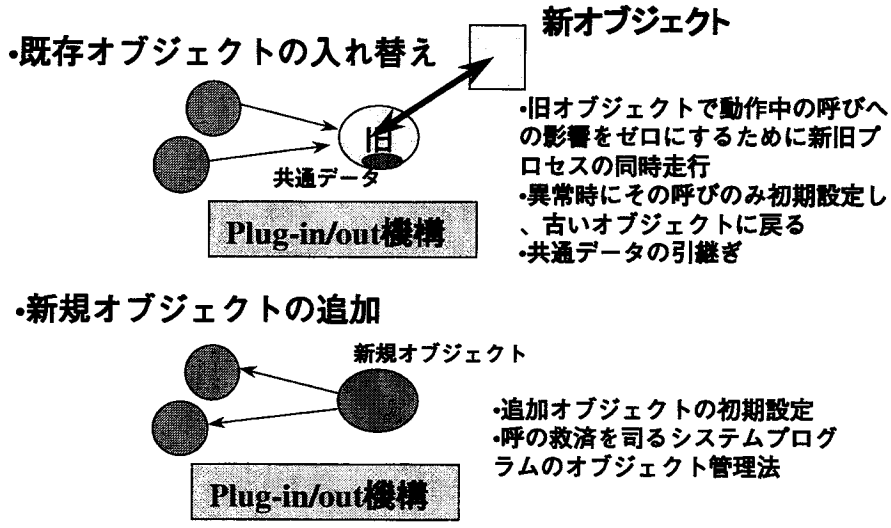


図 18　運用中ファイルの追加・変更モデル

既存オブジェクトを新オブジェクトにより入れ替える場合、旧オブジェクトで動作中の呼びへの影響を無くすために新・旧オブジェクトに対応したプロセスの同時走行を可能にする必要がある。新旧プロセスの同時走行を実現するためには、交換システムにおける通話路系リソースなどの共通データの引継ぎ問題を解決しなければならない。しかも、リアルタイム性を損なうことなく実現する必要がある。新旧プロセス間での矛盾した共通データの引継ぎはシステムダウンなどのシステム異常の原因となる。

また、新オブジェクトにより旧オブジェクトを入れ替えた後、システムに何らかの異常が発生した場合、最も疑わしい新オブジェクトを速やかにプラグアウトする。しかも、すでに新オブジェクトでサービスを提供している幾つかのプロセスを１つずつ解放し、他のサービスを受けているユーザに影響を与えることなく、元の状況に戻るための手段を提供しなければならない。

新規オブジェクトの追加の場合、運用中システムが予期していない新たなオブジェクトの追加が必要であり、そのシステムを管理するシステムプログラムへの新規

オブジェクトの登録、更にはプラグアウト時の登録抹消が自動で行われなければならない。

この様な問題を解決するために、本章では第2章で紹介したプラグイン・アウト単位の明確化を狙いとしたオブジェクト指向技術を前提とした NOSES 制御技術（信頼性を考慮したプラグイン・アウト技術）を提案する。更に、第4章では、本章で提案する NOSES プラットフォーム化技術、第5章で、オブジェクトをプラグイン・アウトするためのネットワーク管理技術、第6章でプラグイン対象となるオブジェクト生成のための開発支援技術を紹介する（**図 19**）。

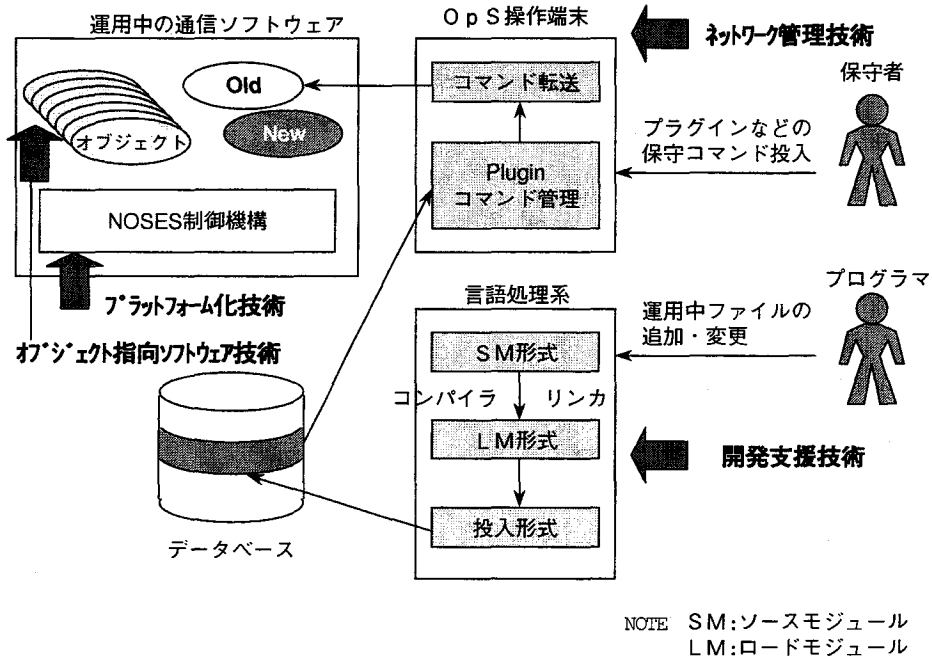


図 19 全体イメージ

3.2.3 システムファイル更新とプラグイン技術

運用中ファイルへのサービス追加・変更手段として従来から行われているシステムファイル更新と、本論文で提案するプラグイン技術がある。

ユーザにサービスを提供している運用中のファイルを追加・変更するためプラグイン機構により新たなオブジェクトが投入された際、ユーザに影響を与えてはならない。矛盾なく追加・変更可能な条件は何か。プラグイン機構が有効な条件を本論文

で明確にする。プラグインで対処出来ない追加・変更については、システムファイル更新によりサービス・機能の追加を行う。

制御系のアーキテクチャとしては、アクト・スタンバイの2重化構成を前提に考えている。プラグインは、ユーザへのサービスに全く影響することなくサービスを追加・変更する事が目的であり、アクト系（現用系）に追加・変更を加える。それに対して、システムファイル更新は、スタンバイ系（予備系）を使って新しいファイルを入れ替える方式である（**図 20**）。プラグインでは、サービスを提供しているファイルに直接追加変更するので、矛盾なくファイルに変更を与えるためには、適用上の条件がある。それに対してファイル更新は、スタンバイ系のファイルを全て入れ替えるため全面的なファイルの変更が可能である。但し、スタンバイ側のファイルをアクト系に切り替えるためには再開処理経由 となるため、長時間保留呼などのある特定の呼のみ救済可能という条件付きとなる。そのため、サービスを提供しているユーザへの影響が大きい（**表 2**）。そため、システムファイル更新は年に1～2回程度の実施に限定されている。しかもユーザへの影響が大きいためにトラヒックの少ない深夜に実施するなど運用面での大きな弊害になっている。

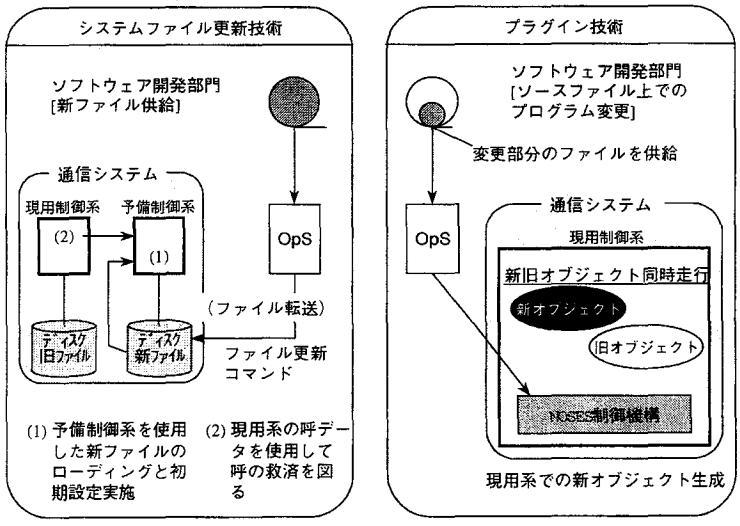


図 20　システムファイル更新とプラグイン技術

表 2　システムファイル更新とプラグインの比較

	プラグイン	システムファイル更新
言語	高級言語	高級言語
アドレス管理	不要	不要
初期設計	不要または部分的に必要	システム再開
サービス中の呼	影響しない	一部のみ救済可能
適用性	制限あり	制限なし

3.3 新オブジェクトのプラグイン

3.3.1 新旧ｵﾌﾞｼﾞｪｸﾄの同時走行と共通ﾃﾞｰﾀの引継ぎ

共通データを持たないオブジェクト

共通データを管理しないオブジェクトの追加・変更から考える。最もトラヒックの大きい電話サービスを変更するケースを取り上げる。プラグイン実行時に複数（大きなシステムでは数千）のユーザに電話サービスを提供中である。従って、電話サービスの閉塞は出来ない。閉塞すれば電話サービスを止める事になり、ユーザへの影響が大きい。変更済みの新電話サービスシナリオオブジェクトをプラグインした後は、新電話サービスシナリオオブジェクトによりサービスが提供される。つまり旧サービスシナリオオブジェクトでサービスを受けているユーザと新サービスシナリオオブジェクトでサービスを受けているユーザが同時に存在する事になる。新旧オブジェクトが共存できる事がポイントになる（**図 18**）。

サービスシナリオオブジェクトを設計する立場の技術者には、**図 18**のオブジェクトモデルを意識すればよい。この NOSES の設計を行う技術者の立場でさらに中を

ブレークダウンする。なお、本論文では、サービスシナリオで共通なデータを管理していないことを想定している。

オブジェクトを具現化するクラスがコンパイルされリンカを通してロードモジュールが作成されて通信システムのメモリ上に展開される。通信システムの建設時やシステムファイル更新時は、加入者データや所データなどの運用データが投入される。それらがメモリ上に、命令部、定数データ部、変数データ部に大きく別れる（どのように分類されるかはシステム依存であるが、本論文ではこの分類を前提に議論を進める)。実際は、コンパイラの作り方、コーディング方法さらにはローダの作りにより何がどこに分類され且つメモリ上のどこに割り付けられるかが異なる。テキストが命令部に割り付けられるのは皆同じであるが、問題は、定数や変数が初期値の有無などによりコンパイラの出力形態が異なる。ローダはこのコンパイラの出力情報とファイル化情報に従って予め決められたルールに従ってメモリ上にデータを割り付ける。また、コンパイラはコンパイラ自身が必要とする定数も生成する。それが何処に割り付けられるか、どのような時に生成されるかもコンパイラにより異なる。NOSES の設計に当たっては、このレベルまでの規定が必要になる。オブジェクトという抽象的なレベルでの設計を進めて行くが、どんな機能を設計するかによりどのレベルまで具体化して設計を進めるかが異なる。その為に通信ソフトウェアの階層化がソフトウェアの生産性向上に如何に効果があるかは第２章で紹介した通りである。NOSES は、システム制御の根幹をなす機能であり、物理メモリへの展開イメージ、カーネルが提供するタスク動作モデルやタスクの実行レベルまで具体化して設計を進める必要がある。しかし、いきなりタスク動作モデルまで落すのではなく、可能な限り抽象化したレベルでモデル化していくことが重要である。

サービス階層オブジェクトのプラグインでは上記レベルのモデルを考慮した実現方式を考えれば済むオブジェクトが、より下位階層のオブジェクトよりもより多く存在する。

新オブジェクト追加時の方式としては２方式ある。間接ジャンプ方式（**図 21**）と直接ジャンプ方式（**図 22**）がある。

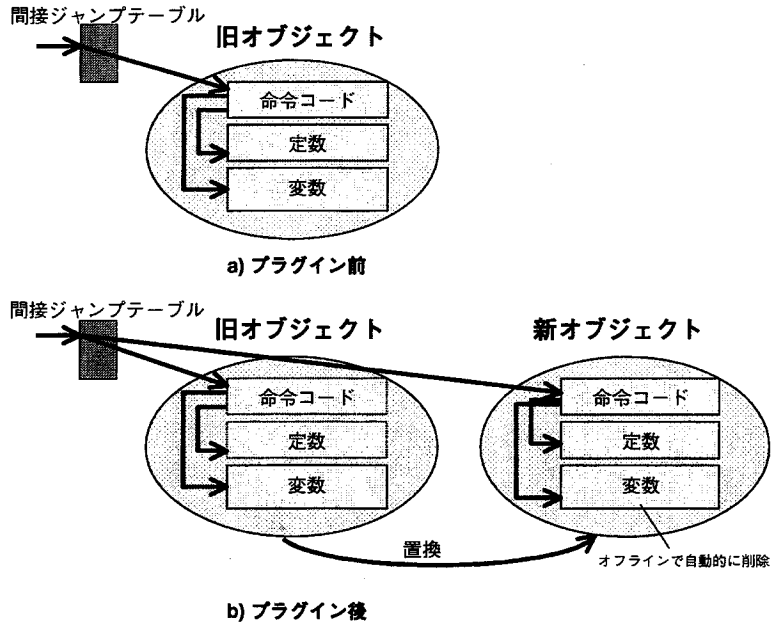


図 21　新オブジェクト追加時の手法（間接ジャンプ方式）

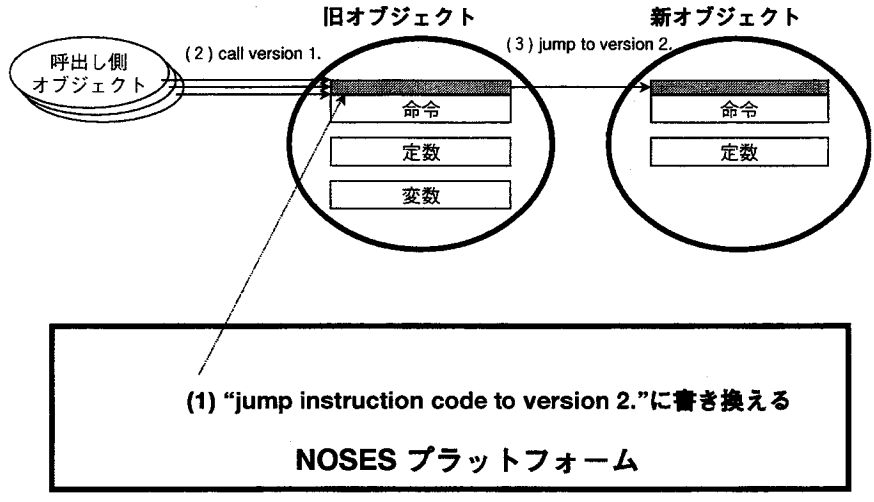


図 22 新オブジェクト追加時の手法（直接ジャンプ方式）

どちらの方式も新旧サービスシナリオオブジェクトが、各々メモリ上に命令部、定数部及び変数部に展開されて２重に設置される。プラグインされる前にサービスを要求したユーザは、プラグイン後も存在している限り旧側の命令部、定数や変数でサービスを受ける。プラグイン後に新たにサービスを受けるユーザは、新しいオブジェクトでサービスを受ける。ユーザの刻々変化する呼データは、変数部で管理される 。サービス階層オブジェクトは、各サービスで共有したり同一サービスでも複数の呼で共有する共通データがないので、全て２重に設置されても問題はない。コードが一箇所でも変更になれば全て入れ替えることになるのでメモリ使用効率の問題が残るが、大変シンプルな構造でプラグイン可能となる。

図 21が示す間接ジャンプ方式は、新オブジェクト追加時、間接テーブルを新オブジェクトの先頭アドレスで書き換える事により実現できる。しかし、通常のオブジェクト間の呼び出しが常にテーブル経由になること、更にプラグイン時に他のサービスと共通に使用するテーブルを直接プラグイン機構が書き込みに行く必要があるため、常に書き込み処理が行われるになる。その為、高信頼性通信ソフトウェアの実現の観点から**図 22**の直接ジャンプ方式が望ましい。

この直接ジャンプ方式は、旧オブジェクトの先頭アドレスに新オブジェクトの先頭アドレスを書き込む方式であり、間接ジャンプ方式の様なジャンプテーブルの共通化が不要である。プラグイン機構が書き込む個所は対象とするオブジェクトのみでありプラグイン機構の動作による異常な処理による影響が局所化出来ると同時に処理効率の面でもメリットがある。

以下、本論文では直接ジャンプ方式を前提に紹介する。

共通データを持つオブジェクト

次に共通データを管理するオブジェクトを考える。交換システムを構成する通話路の管理・制御を司る通話路オブジェクトを例に考える。サービスシナリオオブジェクトとの違いで考慮すべき事は、共通データである通話路の空塞状態を管理している事である。この空塞状態は、局規模に依存する通話路の大きさなどに依存するため所データとして実現される。通話路オブジェクトをオブジェクトモデルで記述すればサービスシナリオオブジェクトと同じ様に表現できる。オブジェクトモデルの利点は、オブジェクトの特性によらずにモデルを統一して表現出来るところに１つの利点がある。このメリットが設計の再利用に繋がる。

新旧通話路オブジェクトを物理メモリ上に展開すると、通話路の空塞データも２重管理になる。新旧オブジェクトが夫々勝手に通話路を解放したり閉塞するとハードウェア上で矛盾を生じることになる。新旧オブジェクト間でこのような共通データを協調しながら制御する必要がある。旧側のオブジェクトに当初からこのような新側との協調機能を持たしておくのは困難であるため、処理矛盾を避けることができない。新旧オブジェクトで所データを共有する必要がある。しかも、旧オブジェクトが意識することなく、自然に新オブジェクトと共有出来る環境が必要である。命令部は、変更があるので新旧で２重になる。定数部も命令部の変更に対応して変わるので２重になる。従って、所データは、変数部に出力し、変数部を新旧命令部で共用出来ることが必要である。

プラグインのオブジェクトモデルは、新旧オブジェクトの２重置きにより表現可能である。物理メモリへの展開は、命令部及び定数部の２重置き、共通データを自動的に引き継ぐことにより実現される。新オブジェクト側で新しく追加になる変数は、旧側と矛盾を生じる事がないためプラグイン可能である。但し、新変数を識別するためにファイル化で工夫が必要となる（ファイル化の工夫については後述）。以上をまとめたのが**図 23**である。

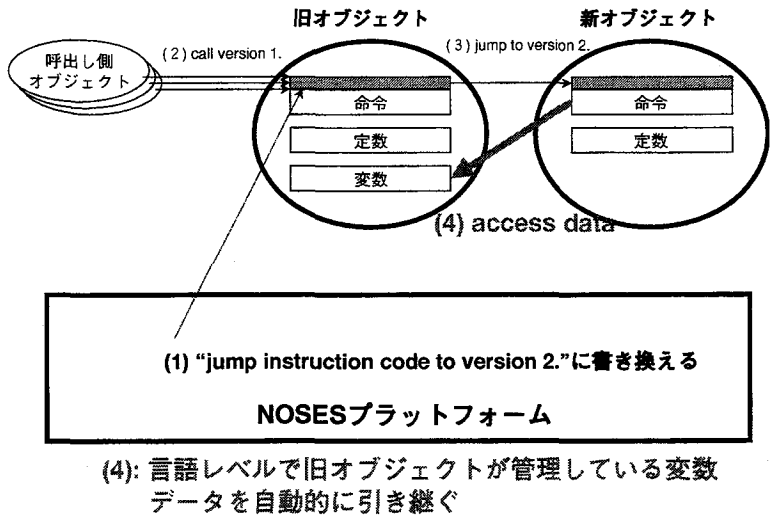


図 23　新オブジェクトの追加手法：共通リソースの引継ぎ

オブジェクト設計者が**図 22**のオブジェクトモデルだけに着目して C++などのオブジェクト言語で記述できる環境条件を整えなければならない。設計者が共通データを旧オブジェクトと新オブジェクトで共有するための仕掛けを意識しなくてもプラグイン可能なオブジェクトの設計が出来るようにする必要がある。

変更したいクラスをエディットしてコンパイルする。コンパイル単位の詳細は本章で後述する。プラグイン用のファイルが作られるときに、旧オブジェクトと共通になる変数部は自動的に破棄される。命令部と既にシステム上にある変数部とのリンクは、オフライン或いはオンラインでも解決可能である。オンライン上での処理時間を短くするのであればオフラインで解決する。夫々のリンクが解決されてメモリ上に展開される（**図 25**）。

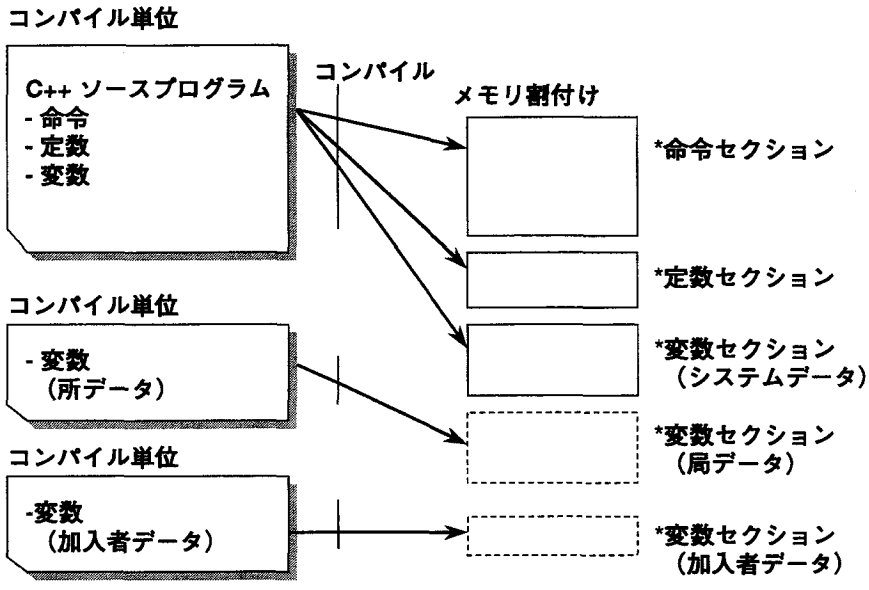


図 24 コンパイラとファイル化条件

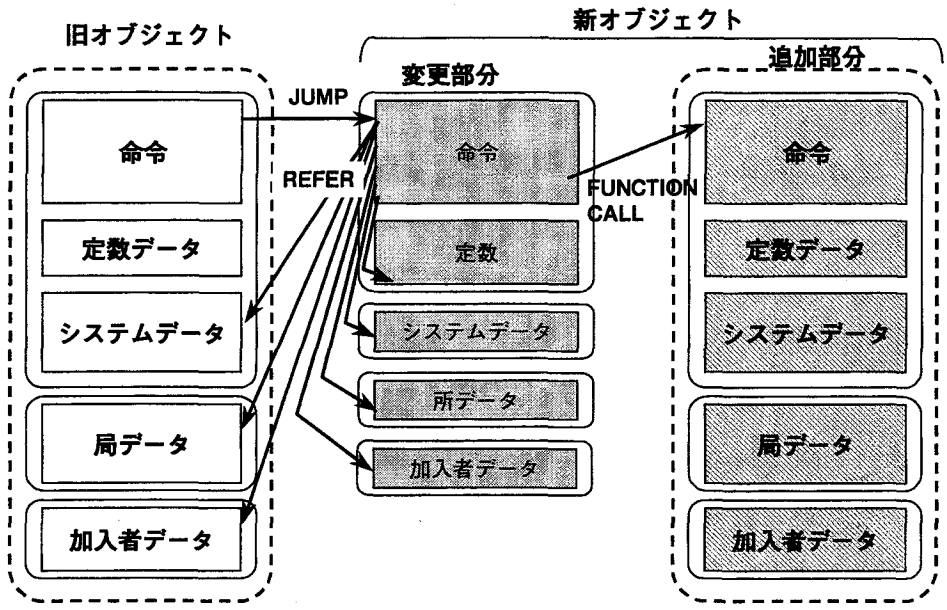


図 25　メモリ上の展開

3.3.2 新オブジェクトのプラグイン機構

次に NOSES プラットフォームの中で、新オブジェクトをプラグインするためのローダ機能を述べる（**図 26**）。

言語処理系でオブジェクトを記述するクラスがコンパイルされ、ある決められたファイル形式に変換されて OpS に送付される（**図 19**）。OpS は、送られてきたファイルの原本を蓄積しておく。次に OpS から通信システムにプラグイン用ファイルが送られる。通信システムと OpS とのインタフェースは、TMN プロトコルで規定された MO で定義されている。この TMN 制御の設計については、第５章で紹介する。通信システム内の TMN 制御で送られて来たプラグインファイルを受信し、一旦、ディスクに蓄える。いくつものプラグイン用ファイルが通信システム内のディスクで管理されている。保守者は、OpS を通して、このディスクに蓄積されたプラグイン用ファイルの一覧表を見ることが出来る。保守者のマンマシンインタフェースは、表形式などの高度なインタフェースが提供されている。そこには追加変更の対象となるサービス或いは機能

名とプラグインオブジェクト名が対応付けられている。以上が全体を通した処理の流れである。

新オブジェクトのプラグインは、既存オブジェクトの変更を行う。システム建設時やシステムファイル更新で投入されたファイルのメモリ割付情報（メモリアドレス）は、言語処理系を実現している開発側で管理可能である。プラグイン用ファイルが対象としている旧オブジェクトのプログラムアドレスをオフラインで決めることが可能となる。つまり、何処のアドレスのファイルが対象であるかを直接 OpS 側から通信システムに通知できる。オブジェクトの追加については、全てリロケーショナル情報によるラベルで扱われる。オブジェクトがプラグインされるメインメモリエリアは、通信システム内で捕捉される。可能な限り制御系に依存する機能は、通信システム内に閉じて実現する事がNOSESの狙いの１つである。

ディスクに格納されているプラグイン用ファイルを引き上げてメインメモリ上にファイルを展開する機能をプラグインローダ（**図 26**）と呼ぶ。このプラグインローダは、オブジェクトをプラグインするためのメインメモリ上のエリアを捕捉する。

ある１つのオブジェクトに着目すると、そのオブジェクトを呼んでいるオブジェクトは複数ある。プラグインの際、プラグインオブジェクトを呼ぶ全てのオブジェクト名を知り、それら全てのオブジェクトに新しいオブジェクト名を通知することは、大規模ソフトウェアほど大変である。これを簡単に実現するために、呼ばれ側で対処する方法を採用している（**図 22**）。プラグイン対象である旧オブジェクトが、他のオブジェクトから呼ばれているエリアに新オブジェクトへのジャンプ命令を自動的に書き込む方式がリアルタイム性と高信頼性を追求する通信システムに最適であることをすでに述べた。

プラグインの契機は、OpS 側からの実行指示である。保守者は、先ほど作成したプラグイン用ファイルの一覧表を見ながらプラグイン開始コマンドを打つ。プラグイン制御が、この保守者の指示に従い、ファイル操作を実行し、ディスク上の対象とするプラグインファイルの格納情報を取得する。この情報をプラグインローダに通知すると共にプラグインファイルをロードすることを指示する。プラグインローダは、エリアの捕捉と新旧の切り替え処理を実行し、終了後プラグイン制御に通知する。プラグイン制御は、保守者に終わった事を通知する。

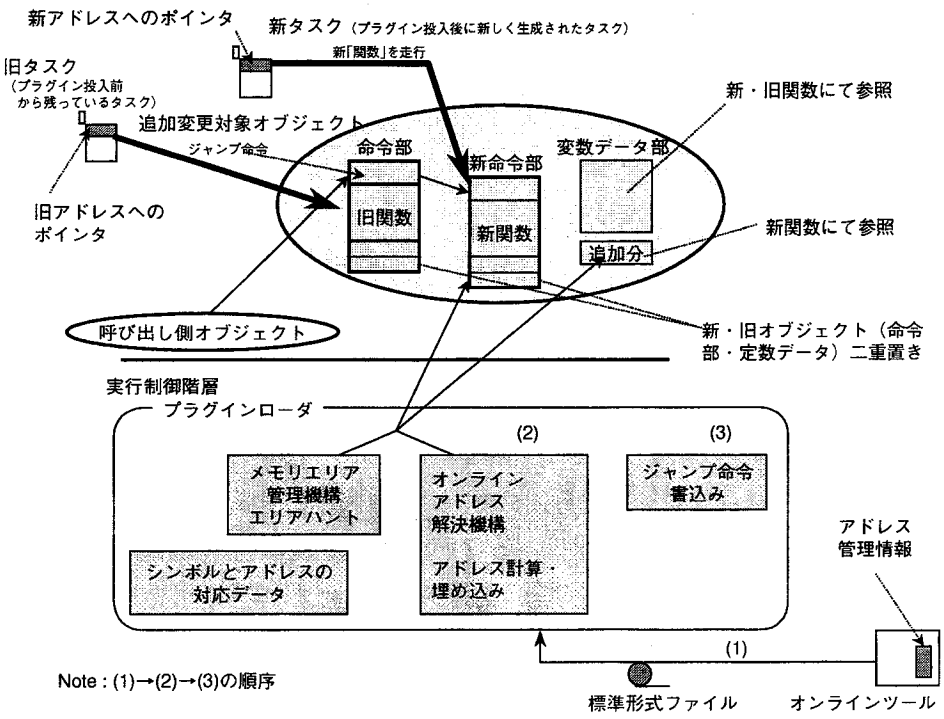


図 26　新オブジェクトのプラグイン機構

3.4 新規オブジェクトのプラグイン

次に新規オブジェクトのプラグイン機構を紹介する。これは、新サービス・機能の追加時に新規オブジェクトの追加に相当する。前述の既存オブジェクトの追加・変更との違いは、新規オブジェクトの追加に伴い、運用中ファイルに存在するオブジェクトとの関係を解決する事である。運用中のファイルを構築した際には、存在していなかったオブジェクトの追加であり当然の事と言える。

3.4.1 オブジェクト間の呼び出し

新オブジェクトのプラグインとの違いに着目した主要課題を以下に示す。

新規オブジェクトに関して、コンストラクタの起動（C++)、呼び出し側オブジェクトへの自分自身の登録と、プラグインの正常性が確認された後の再開管理オブジェクトへの登録などがある。

・新規オブジェクトの初期設定法
・コンストラクトの起動法

・プラグイン正常性の確認とその後の再開管理オブジェクトへの登録法
・新規オブジェクトの呼び出し側オブジェクトへの登録法
・新規オブジェクトが持つ所データの登録法
・（保守）コマンドの登録法

新規オブジェクトのプラグインを実現するためにこれらの条件を満足するにはソフトウェア構造が、上記条件を基本にして構築されている必要がある。単に、プラグイン制御オブジェクトやプラグイン用ローダなどの機能を提供する実行制御を設けるだけでは新規オブジェクトのプラグインは実現できない。ソフトウェア構造自体が、プラグイン可能なソフトウェア構造である必要がある。

運用中のオブジェクトが、プラグインされた新規オブジェクトを呼ぶ際に、この新規オブジェクトがプラグインにより後で運用中に追加されたか、それともシステムファイル更新やシステム建設時に追加されたかを識別することは不可能である。従って、オブジェクト間の呼び出しは、全て１つのルールに従う必要がある。本論文で提案する方式を以下に示す。

・オブジェクトの初期設定を一元管理する（再開管理オブジェクト）
・再開管理オブジェクトにより初期設定される際に自オブジェクトを呼んでくれるオブジェクトに自分自身を登録
・再開管理オブジェクトにより初期設定される際に自オブジェクトで所データがあれば、所データ管理オブジェクトへ登録
・再開管理オブジェクトにより初期設定される際に自オブジェクト内で必要なコマンドを,コマンドを振り分けるオブジェクト（コマンド・メッセージオブジェクト）に登録
・再開管理オブジェクトにより初期設定される際に再開処理オブジェクトへ自分を登録

ソフトウェア構造が、このようなルールに従っていれば、プラグインにより新規に追加されるオブジェクトの数に制限はない。新規オブジェクトのプラグインモデルを**図 27**に示す。

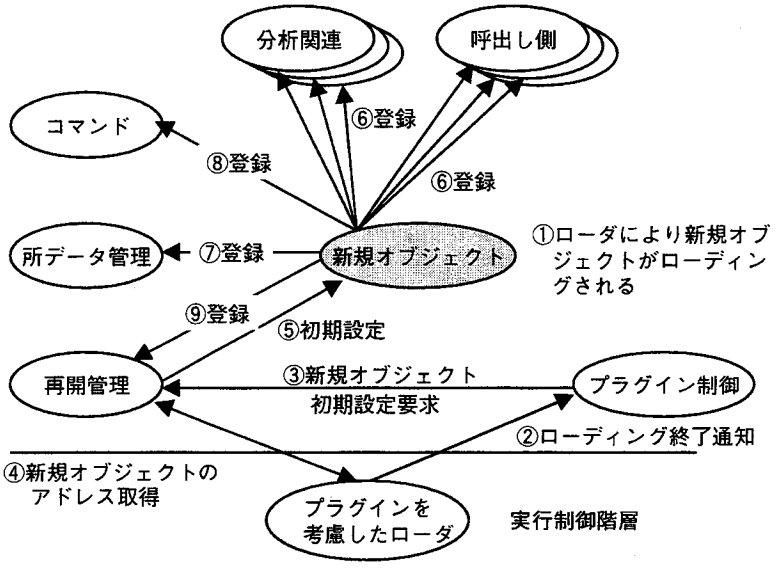


図 27 新規オブジェクトの登録処理モデル

以下の様になる。

- (1)~(2)プラグインローダが新規オブジェクトをローディング後、その旨をプラグイン制御へ通知
- (3)プラグイン制御が再開管理へ新規オブジェクトの初期設定を要求
- (4)~(5)再開管理が新規オブジェクトのアドレスを取得した後、新規オブジェクトを初期設定
- (6)~(8)新規オブジェクトは、関連オブジェクトに登録
- (9)正常終了後、再開管理へ自分自身を登録

3.4.2 オブジェクトの初期設定

オブジェクトを運用中システムのメインメモリ上に展開するまでは、3.3.2 項で紹介した方式と同じである。異なるのは、プラグイン制御オブジェクトがプラグインローダから通知されたオブジェクト名と共に再開管理オブジェクトに新規オブジェクトの初期設定を依頼することである。

再開管理が新規オブジェクトの初期設定を行う。C++言語の適用の場合、コンストラクタを呼ぶ（**図 28**）。C++ではクラスはオブジェクトではないので、この時点でオブジェクトが生成される。

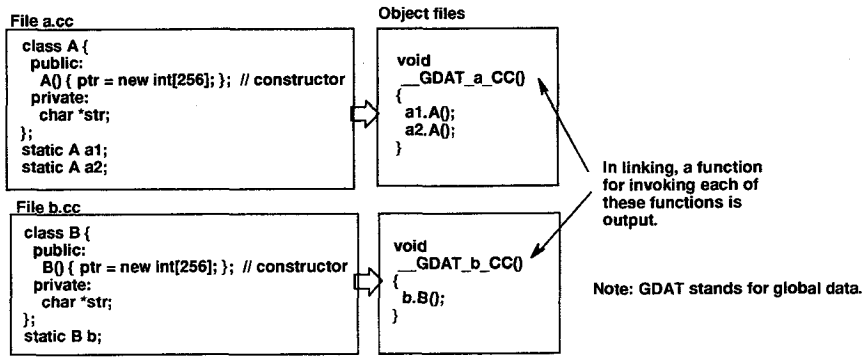


図 28　大域オブジェクトのコントラスタ起動方式

従って、C++で記述されたクラスをプラグインする場合、オブジェクトが生成された後あらためて生成されたオブジェクトの初期設定を行う必要がある。この初期設定により**図 27**で紹介した登録が行われる。

この初期設定方式が、通信システム上で動作する全オブジェクトの初期設定を決める要因となる。

オブジェクト生成に必要なリソースを確保するためにダイナミックにコンストラクトを呼び出さなければならない。クラスその物はオブジェクトではないので、システム環境下でオブジェクトを生成するためにコンストラクタを呼び出す。また、消滅時にはデストラクトを呼ぶ。従って、コンストラクトはオブジェクトを生成するクラス毎に記述する必要がある。スタック上やシステムヒープ上に確保された局所オブジェクトは、実行文中で初期設定すればよいのでさほど問題はないが、大域（グロー

バル）オブジェクトについてはオブジェクトのライフタイムがシステムの運用中永久に存在するため、システム建設時やシステム立ち上げ時にコンストラクタを起動する必要がある。また、C++の場合、ファイル化単位とオブジェクトの対応は自由である。プラグイン時のファイル化（OM 形式）を後述する。ファイル更新時には、ＯＭ形式のファイル化単位はプラグインと合わせる必要がある。

ブートによりローディングされるＬＭ形式ファイルを初期設定するエントリは１つしか作成できない。ブート対象のファイルの中に複数のクラスが存在する場合、そのＬＭ内に各クラスのコンストラクタを呼ぶ為の振り分け機能が必要である。その振り分け機能がファイル初期設定を司る再開管理に呼ばれるインタフェースを提供する。新規オブジェクトのプラグイン時には、プラグインされる新ファイルの中にそれに対応した振り分け機能が含まれていなければならない。この振り分け機能の生成や振り分け機能自身のプラグインは、これらが言語に依存した機能であることからプラグインを利用するユーザに見えない様にツールによる自動生成が必要である。

以上を整理すると再開管理からのオブジェクトの初期設定には以下の３種類が必要である。

- オブジェクト生成
  - 大域（グローバル）オブジェクトを生成するためのファイルに存在するクラス毎のコンストラクタを呼ぶ機能。
- プラグイン初期設定
  - オブジェクト間の呼び出しに必要な新規オブジェクト自身の登録。この登録契機を起動する関数をプラグイン初期設定起動関数と呼ぶ。
- プラグイン終了処理起動
  - 新規オブジェクトをプラグインし、定められた時間監視後に正常である場合、再開管理が新規オブジェクトを呼んで新規オブジェクトのエントリアドレスを再開管理に組み込む。

さらに、このプラグイン監視中に異常が生じた場合、プラグアウトする際に登録したエントリアドレスを削除する必要がある。これを

- プラグアウト処理起動

と呼ぶ。

また、第４章で紹介するNOSESプラットフォーム化を実現する為には、

- エントリ呼び出しアドレスの登録
  - アプリケーションファイルとプラットフォームファイル間などのエントリ呼び出し登録。このエントリアドレス登録契機を起動する関数をスタート起動関数と呼ぶ。なお、この起動関数はプラグイン時には呼ばれない。システム建設、システムファイル更新や再開フェーズで呼ばれる関数である。

が必要となる。

3.5 プラグアウト機構

運用中システムにオブジェクトをプラグイン直後に異常が発生した場合、プラグインの影響で異常が起きたと考えるのが妥当である。そのためプラグイン後、ある時間、例えば２４時間監視し，その期間で異常を検出し際には１コール初期設定を行い、その１コール初期設定の発生回数が許容回数を超えた時にプラグアウト処理を実施することが考えられる。

再開管理オブジェクトでプラグイン監視状態を管理し、その間に生じた１コール初期設定回数をカウントし、ある時点でオブジェクト間の関係を元の状態に戻すプラグアウト処理を行う（**図 29**）。

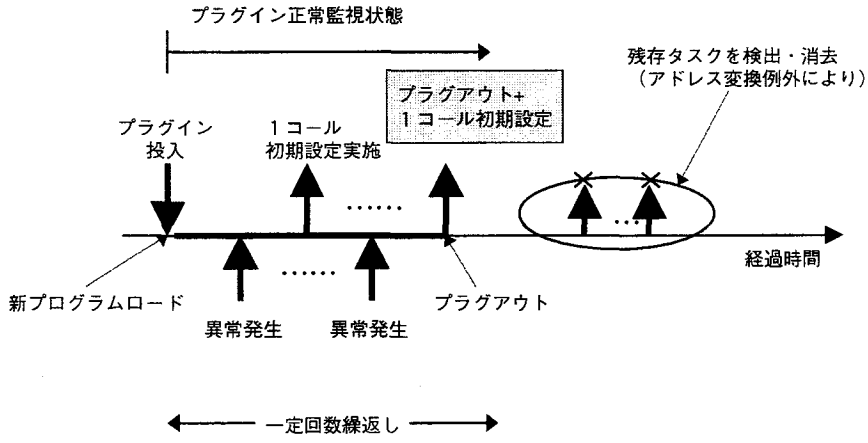


図 29 プラグイン異状時の対策

3.5.1 新オブジェクトのプラグアウト

新オブジェクトのプラグインは、**図 23**及び**図 26**で紹介した様に NOSES プラットフォーム、具体的には実行制御階層のプラグインローダが旧オブジェクトの先頭エリアに新オブジェクトの先頭アドレスを書き込む事により実現している。この一連の処理でプラグインローダが、旧オブジェクトの先頭エリアに書き込まれていた旧命令を旧オブジェクト毎に記憶しておけば、旧オブジェクトへの移行が実現可能である。

プラグアウト処理の要求が再開管理から通知されれば、上述の旧命令を元のエリアに書き込む事によりオブジェクト間の呼び出し関係は旧状態に戻る事が出来る。新オブジェクトでサービスされている“呼”が処理の中断後に再開され、すでに新オブジェクトが消去されている場合には、その“呼”が処理するための実体が存在しないために1コール初期設定が起動するようにする必要がある。これで新オブジェクトのプラグアウトが可能となる（**図 30**）。

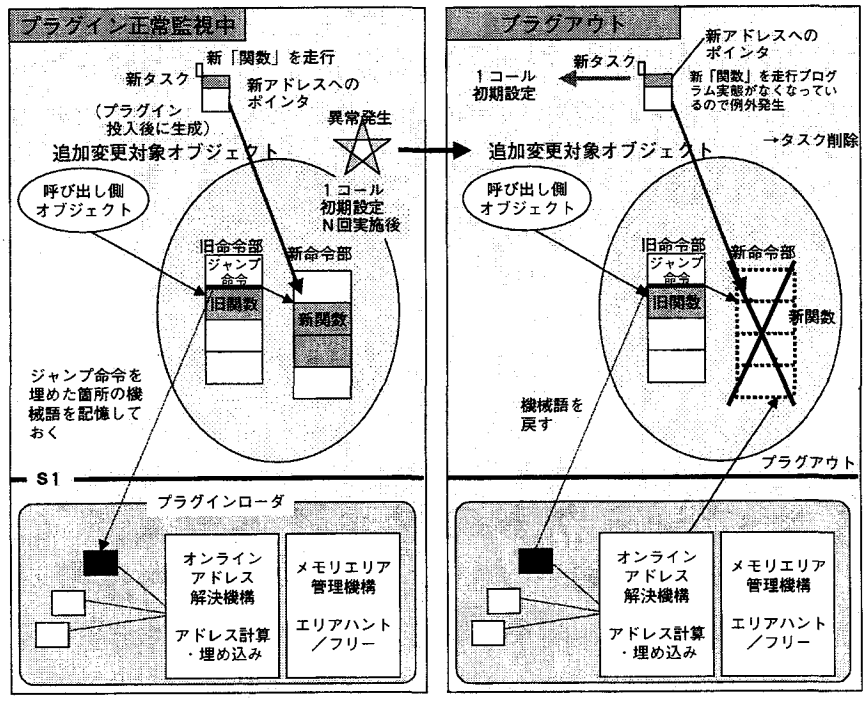


図 30 プラグアウト機構

3.5.2 新規オブジェクトのプラグアウト

新オブジェクトのプラグイン・アウトは NOSES プラットフォームが責任もって処理してくれるが、新規オブジェクトの場合、プラグアウト処理機能を新規オブジェクト自身が提供する必要がある。再開管理オブジェクトからこのプラグアウト処理関数が呼ばれ、周辺の関連オブジェクトに登録したエントリアドレスを解放する必要がある。

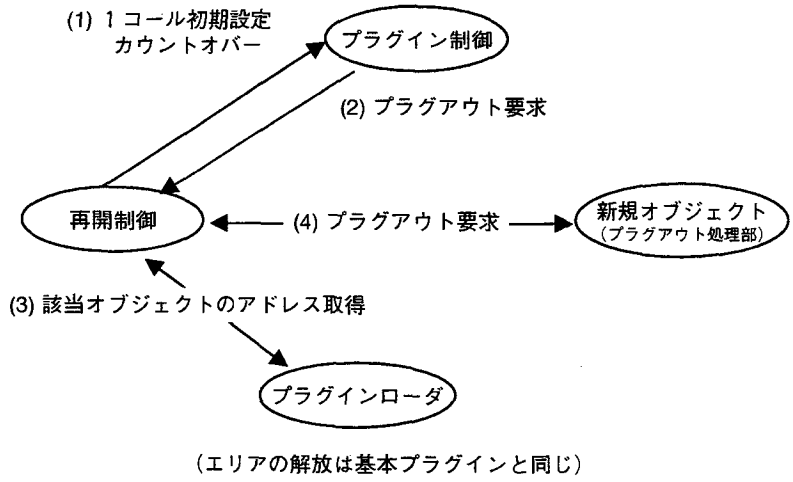


図 31 新規オブジェクトのプラグアウト

3.6 オブジェクトのファイル化

大域（グローバル）オブジェクトについてはオブジェクトのライフタイムがシステムの運用中永久に存在するため、システム建設時やシステム立ち上げ時にコンストラクタを起動する必要があることを前述した。また、C++の場合、ファイル化単位とオブジェクトの対応は自由であり、ＯＭ形式のファイル化単位はプラグインと合わせる必要があることも指摘した。

ブートによりローディングされるＬＭ形式ファイルを初期設定するエントリは１つしか作成でき為に、ブート対象のファイルの中に複数のクラスが存在する場合、そのＬＭ内に各クラスのコンストラクタを呼ぶ為の振り分け機能が必要である。その振り分け機能がファイル初期設定を司る再開管理に呼ばれるインタフェースを提供する。オブジェクトのプラグイン時には、プラグインされる新ファイルの中にそれに対応した振り分け機能が含まれていなければならない（**図 32**）。

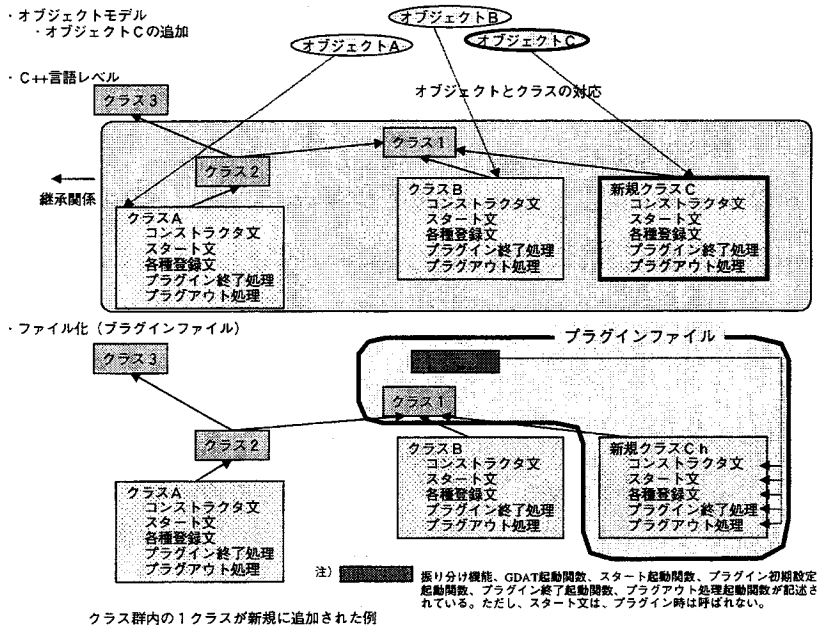


図 32　プラグイン時のオブジェクト，クラスとプラグイン用ファイル化単位

図 32は新規オブジェクトＣをプラグインする場合のファイル化単位から見たオブジェクトとクラスの関係を表している。新規クラスＣはクラス１をスパークラスとして継承することによりオブジェクトＣを生成する。従って、オブジェクトＣをプラグインする為にはクラス１とそれを継承する新規クラスＣ，それに振り分け機能、GRAD 起動関数（コンストラクタを呼ぶ)、スタート起動関数、プラグイン初期設定起動関数、プラグイン終了起動関数、プラグアウト処理起動関数からなるプラグインファイルを作成する必要がある。夫々の関数の機能についてはすでに説明しているのでここでは省略する。

プラグイン時のファイル化単位の詳細な工夫については0に紹介する。

3.7 評価

3.7.1 静的評価

通信ソフトウェアがプラグイン・アウトの適用に対して理想的に構築されていれば、本項が示す静的な評価がそのままプラグイン・アウトの適用評価となる。しかし、実際の通信ソフトウェアは、その様に構築されることはまれである。実際には通信ソフトウェアの動的な振る舞い、即ちオブジェクトの動的特性を性格付ける OS が提

供する実行環境とその利用法に左右される。後者については次の項で評価することとし、本項では理想的な場合のプラグイン・アウトの適用評価を述べる。

本論文で提案する新規オブジェクトのプラグインに関する制約は基本的に何もない。プラグイン適用の制約は、既存オブジェクトの変更に対する限界により決定される。既存オブジェクトの入れ替えで既存の変数、例えば通話路の閉塞管理は既存オブジェクトから引き継ぐ必要がある事をすでに述べた。従って、既存オブジェクトが管理する変数が持つ構造を変更するような新オブジェクトのプラグインは実現できない。この制約がプラグイン適用性評価を決定する。

NTT の過去数年間で実施されたバグ対処のためのパッチ処理（基本プラグインに対応）とサービス・機能追加（応用プラグインに対応）時の静的な特性を調査した結果を**図 33**に示す。

バグ対処では、命令部の追加変更と定数の追加変更でのみ対処可能である。従って、本論文で提案するプラグインが 100%適用可能である。サービス・機能追加では、25%の項目が変数の変更を必要としている。従って、本論文で提案するプラグインが 75%適用可能である。適用できなかったサービスの例としては、明細料金サービスや閉域サービスの様に当初予期していなかった加入者データ構造の変更が必要になったケースである。

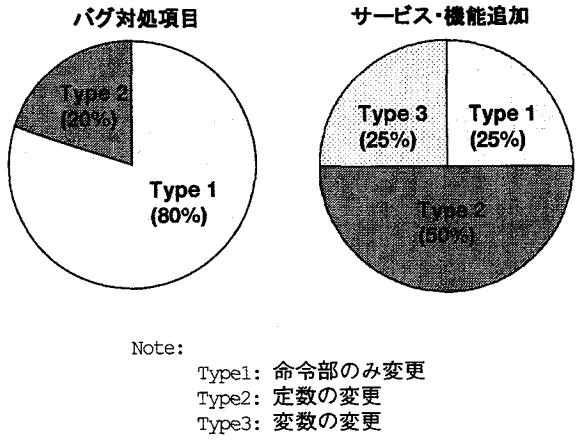


図 33 既存オブジェクトの変更に関わる評価

3.7.2 動的評価

オブジェクトの実行環境は、実行制御階層のカーネルが提供するタスク【31】を活用する。そのタスクが提供する実行環境下で、C++で書かれたオブジェクトのメンバ関数がメンバデータをアクセスしながら他のオブジェクトにサービスを提供する。

オブジェクトがカーネルのタスク環境下で動作する。必要であればオブジェクト自身がカーネルの提供するシステムコール（CRE_TSK など【31】）を利用する。例えば、Caller サーバオブジェクトがタスクを生成し、このサーバ環境で動作するサービスシナリオオブジェクトによりユーザにサービスを提供する呼処理モデルが考えられる。このモデルは、第2章で提案しているモデルに相当する。オブジェクトモデルがほぼ出来上がった時点で、その裏にあるタスクモデルとの対応を考える必要がある。上記に挙げた例では Caller サーバオブジェクトがアクティブオブジェクト、サービスシナリオオブジェクトがパッシブオブジェクトに対応する。アクティブオブジェクトは、自分自身で実行環境を持っているので自律的に動く事ができるが、パッシブオブジェクトはアクティブオブジェクトに従属して動作する。

タスク環境／タスクモデルを言語とは独立にカーネルが提供する。従って、アクティブオブジェクトやパッシブオブジェクトに関わりなくそのメンバ関数は、カーネルが提供するタスクで動作する。アクティブオブジェクトのメンバ関数もカーネルが提供するタスク環境で動作する。プラグイン機構から見るとメンバ関数がアクティブオブジェクトかパッシブオブジェクトであるかには依存しない。

プラグイン機構から見て重要な事は、メンバ関数がその処理の中で処理を中断するか否かである。中断している間にプラグインされると旧メンバ関数で動作していたタスクが、中断点から再開した時に再度旧メンバ関数側で動作する。勿論、プラグイン後に新しく生起したタスクは、新メンバ関数が走行する。新旧同時走行が可能になる。

時刻 X 時に中断していた旧タスクが起動される。その時すでに新オブジェクトがプラグインされているが、旧タスク上のスタックに旧オブジェクトの中断点再開アドレスが格納されているので、旧オブジェクトが走る。ここにプラグイン適用時の限界がある。**図 34**に課題例を示す。そのために静的にはプラグインが実現可能であっても、動的には困難となるケースが生じる。

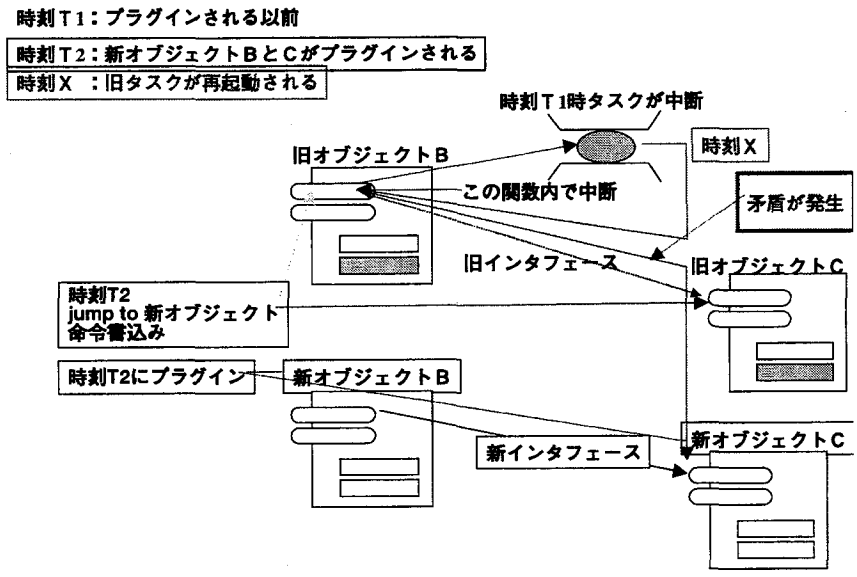


図 34 ダイナミック特性の課題例

上図は、新オブジェクト B のプラグイン時、オブジェクト 間インタフェースが変更になった場合を示している。

複数のオブジェクトにまたがった変更をプラグインで実現するには、それら複数の新オブジェクトをローディングした後それらを同時に旧側から切り替える必要がある。そうすれば、オブジェクト間インタフェースが変更になっても基本プラグインを実現できる。しかし、中断がある場合、中断点から再開されたタスクは、旧オブジェクト B を実行する。ところが、旧オブジェクト B が旧オブジェクト C を呼ぶとすでに新オブジェクト C に切り替わっている。旧インタフェースで新オブジェクトをアクセスしてしまう為に処理矛盾が発生する。この矛盾を防止する技術を本章末にある 3.8.2 に紹介する。技術の鍵は、(1)C++のメンバ関数呼び出しを全て間接呼び出しにする、(2)タスクにタイムスタンプを付けて管理する。スタンプを見て新旧どちらのオブジェクトを走らせるかを逐一判断する， (3)(1)と(2)を常に実行すると間接呼び出しが２段になり非効率となる。そこで旧タスクが無くなった時点で間接呼び出しを１段に切り替えることで対処できる。しかし、本方式は相当複雑な処理となる。コンパイラへのプラグイン専用機能の盛り込みや処理時間を節約するために特別な制御系の

工夫が必要になる。また、カーネルのタスク管理の変更も必要になる。そのため、本論文では中断があるメンバ関数を含むオブジェクトがあり、かつオブジェクト間インタフェースが変更になる場合は、プラグインの対象外とすることで評価を行っている。

図 34は、通信ソフトウェアの動的振る舞いの１例である。動的な振る舞いに対する評価を行う為に動的な振る舞いを分類すると**図 35**の様になる。

先ず、オブジェクトを閉塞できるか否かで分類する。即ち、閉塞出来ればオブジェクトプラグイン時新たなイベントの受付を止め、全てのイベント処理を終了した時点でオブジェクトのプラグインを実施できるため、上記で説明した中断したタスクが存在しない。従って、閉塞できればプラグインの適用が可能となる。

次に、閉塞出来ないオブジェクトの場合、その一連の処理で中断処理が存在するか、否かで分類する。中断処理が存在しないような通信ソフトウェア構造にする事によりプラグイン適用範囲を広げる事が可能である。

中断処理がある場合、オブジェクト間インタフェースの変更があるか否かでプラグインの適用が分類される。

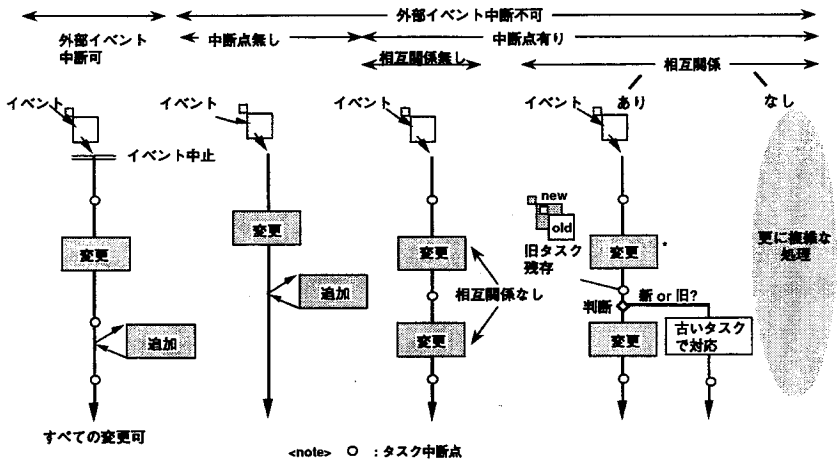


図 35　通信ソフトウェアの動的振る舞い

分類された動的振る舞いと静的な振る舞いを組み合わせた評価表（**図 36**）を作成した。

動的特性 / 静的特性		サービス中断OK	サービス中断NG			
			タスク中断無し	タスク中断有り		
				相互関係無し	相互関係有り	
					新旧の判断	対策無し
無制限		11	12	13	14	15
局データ	追加	21	22	23	24	25
	変更	31	32	33	34	35
加入者データ	追加	41	42	43	44	45
	変更	51	52	53	54	55
インタフェースの変更		61	62	63	64	65
メモリ常駐タスク		71	72	73	74	75
初期必須		81	82	83	84	85
データ構造の変更		91	92	93	94	95
その他(新規ハードの追加)		101	102	103	104	105

図 36　プラグイン適用評価表

評価表でメッシュ部分がプラグイン適用可能な範囲を示している。

このプラグイン表を使って NTT の過去数年間でサービス・機能追加した項目を分類した結果を**図 37**に示す。

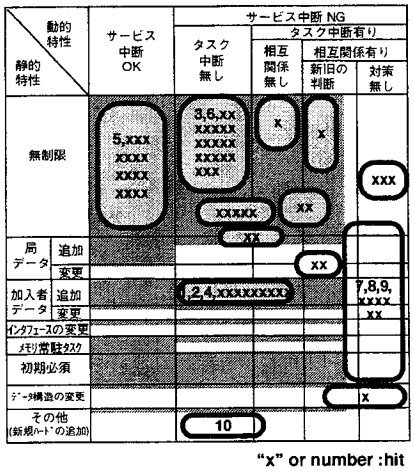


図 37　プラグイン適用評価

この評価から動的な特性を考慮すると 60%のサービス・機能の追加がプラグインで可能であることがわかる。具体的なサービス追加例を**図 38**に示す。

	項目	*	内容
(1)	ダイヤルイン	x	サービスシナリオの追加
(2)	発 ID	x	サービスシナリオの追加
(3)	トラヒックカウンタの追加	x	運用処理の変更
(4)	トーキの追加	x	サービスシナリオの変更
(5)	警報の改良	x	運用処理の変更
(6)	パーソナル番号計画	x	サービスシナリオの追加
(7)	明細料金	-	加入者データ構造の変更及び大幅な処理の変更
(8)	閉域サービス	-	同上
(9)	ISDN 共通線	-	同上
(10)	三者通話	-	同上

<*> x : 適用可能; - : 適用不可

図 38　サービス追加評価

3.8 結言

前章で提案したオブジェクト指向通信ソフトウェアを運用中の通信システム上で着脱可能とするプラグイン・アウト技術を提案した。これによりサービスを受けているユーザに影響なく運用中の通信システムにサービス・機能の追加･変更が可能になる。リアルタイム性と高信頼性を踏まえた上でオブジェクトの追加･変更ができることにより，従来のシステムファイルによるシステム立ち上げを待たずにサービス・機能の追加ができることから，今後の通信システム構築における必須機能と考える。

本章で提案した技術は，所内試験で評価され，その後，1996 年 12 月に導入された大容量 PHS で実際に運用で評価された。その結果，導入後 2 週間でバグフィックスを含めた機能追加・変更 200 件（メモリ規模で数メガ）に何ら問題なく適用でき，

提案したプラグイン機構の信頼性と有効性，ファイル管理性の容易性、さらにサービス・機能追加の即応性が確認された。従来の方式では，この様な短時間でこれだけの規模の追加・変更は困難であり，過去に実施した経験がない。大きな変革として評価できる。

付録

3.1 プラグイン時のファイル化単位の工夫

既存オブジェクトの変更を実現する場合、そのオブジェクトのすべてのプログラムを変更することはない。限定されたメンバ関数の変更や追加で済む。メモリを経済的に使いたいと考えると、その変更・追加分だけを自動的にプラグインしてくれると都合がよい。プラグインしたい人が、追加・変更したいオブジェクトをコーディングし、プラグインコマンドを入れると、後はツールがオブジェクトに対応したC++のクラスをコンパイルし、既存のオブジェクトに対応したOMとを比較し、差分だけをプラグインロードにローディングすればメモリ効率が図れると同時に、プラグインの効果を満足できれば最も効率が良い。

しかし、差分がとれるためには追加・変更したコードが正確に把握できる必要がある。通常のコンパイラは、このような条件保証をしていない。従って、たとえ、一つのメンバ関数の変更でも命令部と定数部の二重化は避けられない。

新オブジェクトがプラグインされた後、その新オブジェクトに再度修正が必要なことがある。三度の追加・変更でメモリが3倍になるようなことは避けたい。問題点と対策を**図 39**に示す。この図ではクラスが一つのプラグイン単位として説明されているが、今まで紹介したように、システム条件によっては一つのクラスが幾つかのファイルに分割される。原則、クラス単位としている。プラグインファイルは、一つのプラグイン作業単位に必要なOM形式のファイルの集合体である。

メモリ管理の効率化を図るために実行制御のプラグインエリア管理で不要となったエリアを積極的に解放する機能を盛り込むことが得策である。ただし、汎用的なガーベッジコレクションのようなエリア管理方式は、実時間処理システムには不向きである。そのため、1度目のプラグインされたエリアを解放するが、解放されたエリアを集めるような処理（ガーベッジコレクション）の提供は NOSES では対象外にしている。したがって、空エリアの虫食い現象が生じる。この虫食い状態は、1年間に何度か実施されるシステムファイル更新できれいにされる。したがって、メインメモリの大きさ、プログラムの変更頻度とその範囲，そしてシステムファイル更新周期などを考慮して効率的なメモリ管理方式を考える必要がある。

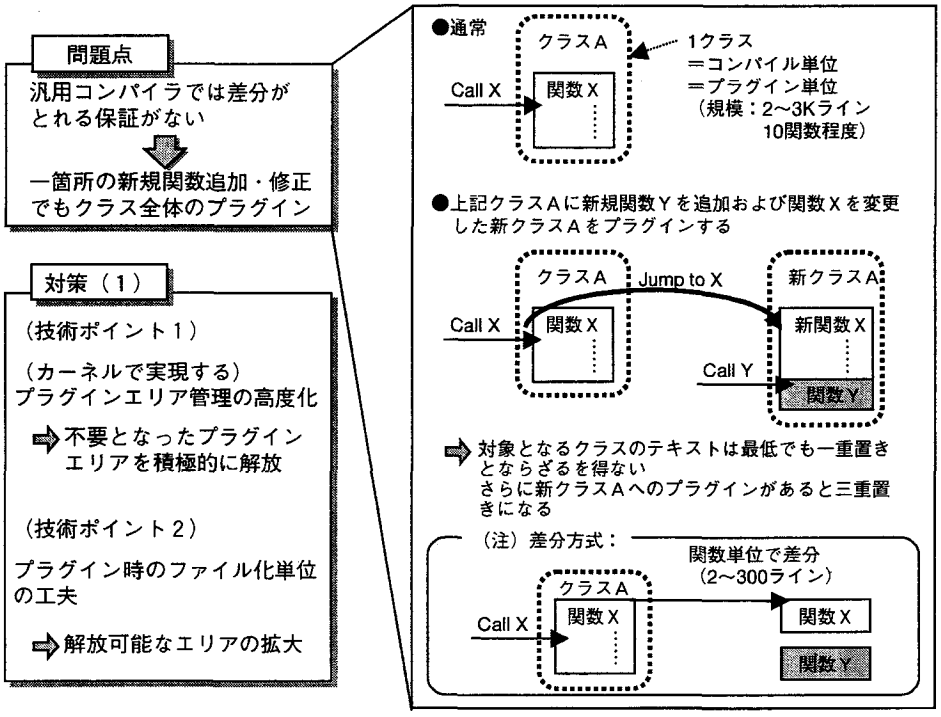


図 39　メモリ使用効率向上対策の必要性

図 40がプラグインエリアの管理を表している。

前のプラグインで追加された新オブジェクトに新関数があると、次のプラグインでもその新関数があるかぎり、一つ前のオブジェクトを解放できない。プラグインは、ファイル更新などでローディングされた最初のオブジェクトにジャンプ命令が自動的に挿入される。一度目のプラグインで追加になった関数へのジャンプ命令は、二度目のプラグインでは挿入されない。したがって、二度目のプラグインでは、この場合、一度目のオブジェクトを解放できない。そこで、一度目のプラグイン時に、新メンバ関数が追加になるときは、その関数を別ファイルにしておけば、二度目のプラグイン時に、新関数部分を残して一回目のオブジェクトを解放可能となる。

図 41は、上述のファイル化に関する工夫の仕方を示している。

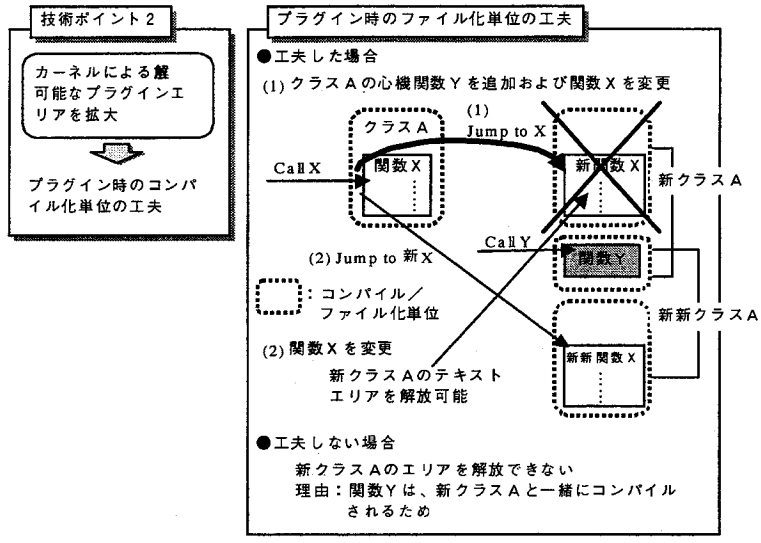


図 40　プラグインエリア管理の高度化

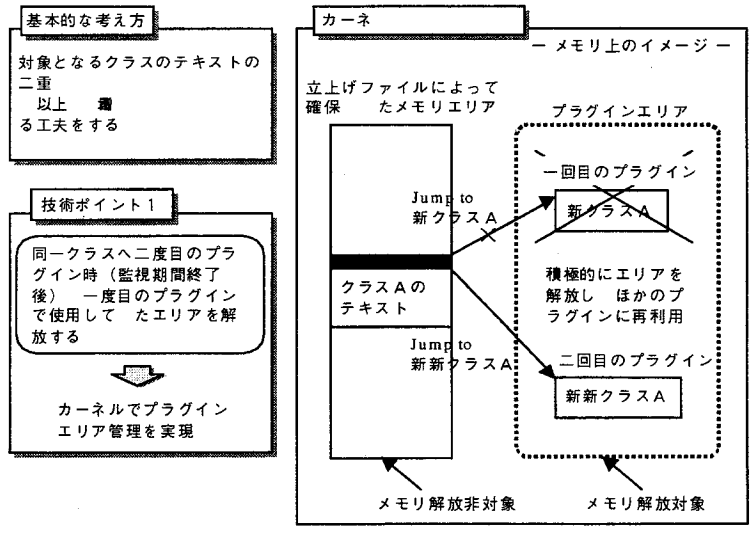


図 41　プラグイン時のファイル化単位の工夫

3.2 中断があるソフトウェアの対策

交換システムは、膨大な数の呼や保守運転コマンドなどのジョブを同時に処理する必要がある。ところが、一つのジョブが始まりから終了まで CPU を専有してしまうことはできない。これらジョブは実時間性が要求されるので、自分が専有してしまうと他のユーザの処理受付けがそれだけ遅れることになる。少なくとも、受付けを遅れることなく、各ジョブを多重で行う制御が必要となる。このような状況から、交換システム内の各ジョブは、自分の処理を中断・再開（中断点に戻りながら）しつつ進めていく。

中断・再開を伴いつつ処理を行う実体（コンテキスト）は、実行制御階層が提供するタスクを用い、生成・中断・再開・消去などの制御により処理を進めていく。ここで、中断のあるプログラムに対して、プラグインによる変更をかけた場合の動作について考える。交換システムに入るジョブ、特に通常の電話については休みなしに発呼があり、かつその受付けを止めることなどは許されない。すなわち、プラグインによりプログラムを変更しようという時には、常に処理中のジョブがあり、当然、中断しているタスクも多数あるということになる。

もし、プラグインによりプログラム修正の箇所がその電話呼処理のメインルートの一箇所であれば、プラグイン箇所以降の処理に矛盾を起こすようなことをコーディングしていない限り問題はない。矛盾を起こすコーディングはバグであり、試験期間中に取り除かれる。次に、このメインルートの複数箇所にプラグインをあてたらどうなるかを考える。全プラグインがある一連の無中断の処理の流れの中に入るなら（これをスレッドと呼ぶ)、これらの処理を一括してクロック割込みマスク状態で行うので、問題なくプラグイン可能である。それでは、この複数のプラグインが１スレッドではなく、複数スレッドに対してなされる場合，言い換えると、タスク中断があるルートにプラグインする場合が課題となる。

この場合、プラグイン実行直前にマスクをかけて排他制御をやったとしても、中断部分のタスク待ちキューにタスクが残っていることになる。すなわち、プラグインによってプログラムが変わったとすると、中断キューに残っている旧いタスクは自分の知らないうちにプログラムが新しくなってしまい、処理矛盾の可能性が生じる。

たとえば、下の例でプログラム１と２を同時に変更したとする。

→プログラム１→（中断Ａ）→プログラム２→（中断Ｂ）→プログラム３→

プログラム１と２の変更にあたってはマスクをかけて新しい入力を止めて行うが、中断Ａの部分にタスクが残っている。その残存タスクは旧プログラム１の処理を通っているが、該タスクが中断キューＡより起床したとき、プログラム２は新プログラムに変わって（正確にいうとプログラム２の中のサブルーチンが変更になっている場合に）いるので、そのタスクは自分がどうしてよいのか分からなくなることがある。

具体的な例を示す。プログラム１とプログラム２があるプロトコル処理を行っているとし、両者をまとめて変更するプラグインを実施した際、それらが持っている「プロトコル状態変数」の意味を変えたいことがある。もともとプロトコル状態変数が０＝空、１＝応答待ち、２＝APL 処理待ちというような状態を持っていたところに、０＝空、１＝データ送出待ち、２＝応答待ち、３＝APL 引き渡し待ち…などのような変更をしてしまうと、旧タスクが認識している状態変数の意味が新プログラム上で変わっているために、中断Ａ以降の処理がまともに進まなくなる。システムダウンの可能性も十分に考えられる。

このように中断があるものに対しては、一定期間入力を止めて（サービス閉塞）、中断キューにあるタスクの処理が全部終わるまで待てばよい（掃き出し）が、これでは絶えずサービスの要求がある電話基本呼サービスに対しては適用できない。保守コマンドのように、一定期間入力を差し控えても問題の無いルートや、呼処理でも新規サービス追加でかつ既存部分にほとんど変更がない場合（分析のケースを一つ増やす程度）なら適用可能である。

適用不可能なケースへの対応策として考えられることは、新・旧タスクがそれぞれ別々のルートを走行して、プラグイン後に生成したものだけ新ルートを走るようにする考えがある。新旧同時走行を実現するには、完全にプログラム・データを二重置きにし、タスクは自分がどちらを実行すべきかを判断する機構が各関数の先頭に必要となる。タスクにタイムスタンプをつけることにより新旧が判断でき、新タスクは新プログラムを、旧タスクは旧プログラムを走ることになり、矛盾の発生が回避可能となる。

図 42にこの方式の説明を示している。

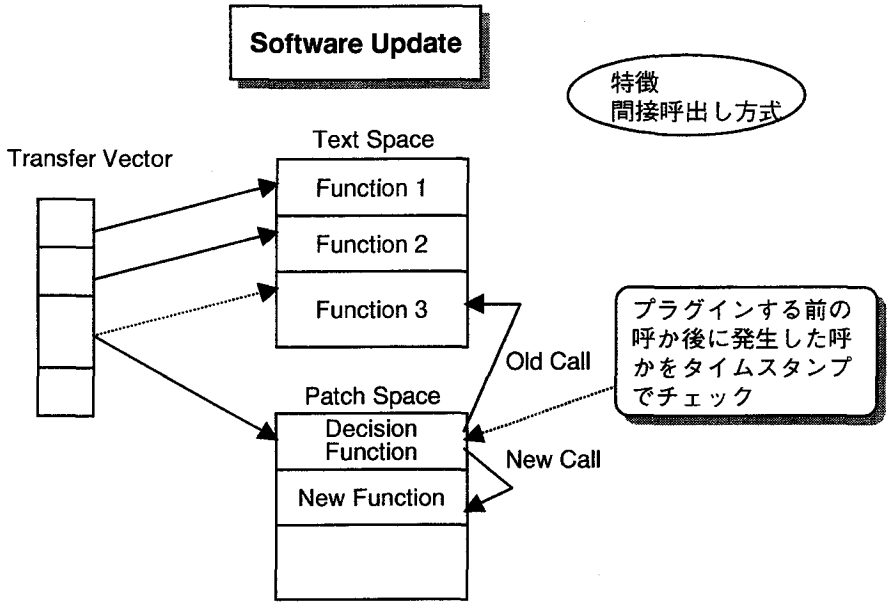


図 42　スタンプを用いた実現方式

上記方式は、すべての関数呼出しを間接呼び出しで実現している。これを Transfer Vector と呼んでいる。かつタスクごとにタイムスタンプを管理し、その呼がプラグイン前の呼か否かを管理する方式である。プラグインする前の呼であれば、旧関数を選択する。これで先ほど述べた処理矛盾がなくなるが、処理時間がかなりかかることになる。その対策としてコンパイラとプロセッサに工夫をして専用化するなどの方式が考えられる。

NOSES では汎用のプロセッサとコンパイラを前提とした方式を提案しているので、上記のような専用化が必要な方式は，本論文では対象外としている。このあたりを考慮して、複雑なサービスはファイル更新にまかせ、プラグインは完璧なバグ対処と軽微なサービス追加を狙っている。

本論文が提案する方式でサービスを矛盾なく追加するためにいくつかの方策が考えられる。中断がない場合、およびサービス閉塞が可能な場合は上記に述べたとおりであるが、さらに次のようなガイドラインにより対応することが考えられる。

・頻繁に発呼のある呼処理ルートでも、プラグインコマンド自体は最下位タスクレベルで実行されるので、それより上位のレベルのタスクの中断を含んでいても可能である。(上位のキューは掃き出される)。しかし、起床待ちキューにタスクが残っている場合には、(実行キューではない)、掃き出しできないので一般に不可となる。このような場合には、新プログラムの作りが冗長になるが、旧プロセスが新プログラム（プラグイン後の新プログラム）を走行しても問題がないよう新旧 OR 条件でコーディングすればプラグイン可能となる。
・ある関数の中でプラグイン監視時間（たとえば 24 時間）を越えて存在するようなタスクがある場合、プラグインできない。これは、監視時間終了で旧オブジェクトを消去するためで、旧タスクがまだ残っていたとすると旧オブジェクトがなくなってしまうとアドレス変更例外が生じる。一般には、次に述べる常駐タスク以外はこのような作りはしないので、もしも必要な場合には（ある関数の中に中断しながら無限にループするような作り)、サービス閉塞ないしサービス強制切断などの処理（外部からのイベントを閉塞、タスク消去、新オブジェクトプラグイン、新タスク生成という手順）が必要となる。呼処理を閉塞しないという前提があるので、これは最後の手段となる。
・外部イベント監視、プロトコルマシンのように常駐のタスクがある場合は、このタスクと同じファイル化単位内にあるプログラムは適用不可能である。なぜなら、常駐タスクごとプログラムを変更しても実際に動いている旧タスクのポインタは旧プログラムを指しており、新プログラムに移行しないためである。この場合には、「周期監視を実現する無限ループ部分」と「分析」や「処理実行」などのアルゴリズム部分を別関数として別ファイルになるように設計することが必要である。これにより、無限ループ部分以外（大部分）はプラグイン対処可能となり、プラグイン適用範囲が広がる（図B)。
・サービス閉塞の例は、前に示したように保守運転処理に適用性がある。しかし、(保守）コマンド処理の変更は保守コマンドを入れないことにより簡単に対応できるが、保守系でもメッセージ処理やトラヒック観測のように、保守者ではなく交換システムの内部的なイベントにより起動されるものがある。この対処としては、そのようなイベントを閉塞できるようにしておく必要がある。自律メッセージルートならディスクリミネータでメッセージの発生を止める、あるいは、まずイベントの発生を止めるための仮の処理をプラグイン（イベントを廃棄する）で実現し、次に実際に行いたい変更をし、再度イベントを拾う処理をプラグインにて回復するなどという手法がある。これは上記(2)の変形ともいえる。

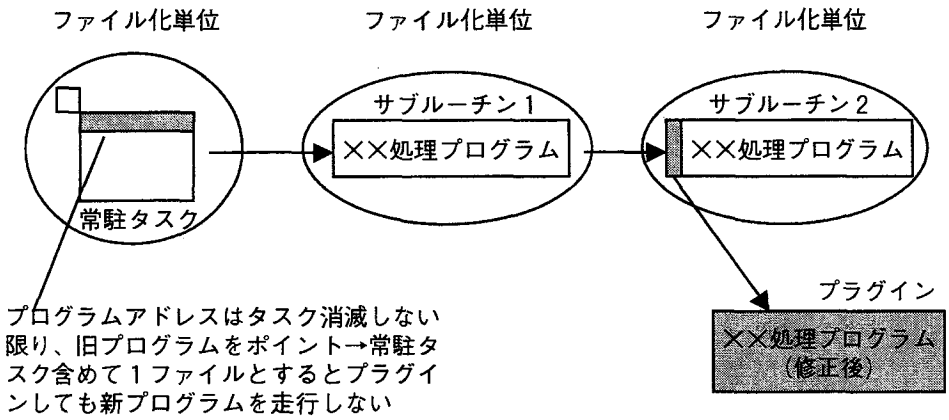


図 43　無限ループを含むプログラムのプラグイン適用例

第4章プラットフォーム構築技術

【関連学術論文【A2】【A3】【A5】】

4.1 緒言

本章では，第3章で提案したプラグイン・アウト機能を多種多様な通信システムに適用するための共通プラットフォーム化技術を提案する。更に，システムの分散化をサービスプログラム設計者に意識させない言語レベルの分散OS技術についても提案する。言語モデルが提供するタスキング機能をシステム分散制御にまで発展させることにより，サービスプログラム設計者は，タスク間通信を提供する各種システムコールを意識することなくサービスプログラム設計を行うことが可能となる。

4.2 プラットフォーム構築

4.2.1 階層化

第2章でプラグイン単位を明確にするためにオブジェクト指向通信ソフトウェア構成法を提案し、第3章でそのプラグイン・アウト技術を紹介した。これらの技術を基本としたソフトウェアの階層化を**図 44**に示す。

NOSES 制御技術を提供するプラットフォームは、色々なアプリケーションそしてシステムに共通に使えることが重要である。今後のサービス需要は、不透明である。これに柔軟に対応するために土台となるプラットフォームを構築する必要がある。この様な階層化されたソフトウェアでは階層毎にファイルを開発・管理することが可能となる。

例えば、次の様な単位で技術者をグループ化／階層化する事が可能になる。実行制御ファイルを管理するグループ、プラットフォームを管理するグループそして各アプリケーションファイル毎（プロダクト毎に対応）に管理するグループである。尚、リソース階層は、通信システムの論理リソース化によりアプリケーションプログラムの作りやすさを狙った階層であり、この階層はアプリケーション毎に個別になるので、アプリケーションファイルと一緒に管理する、という様な体制が考えられる。

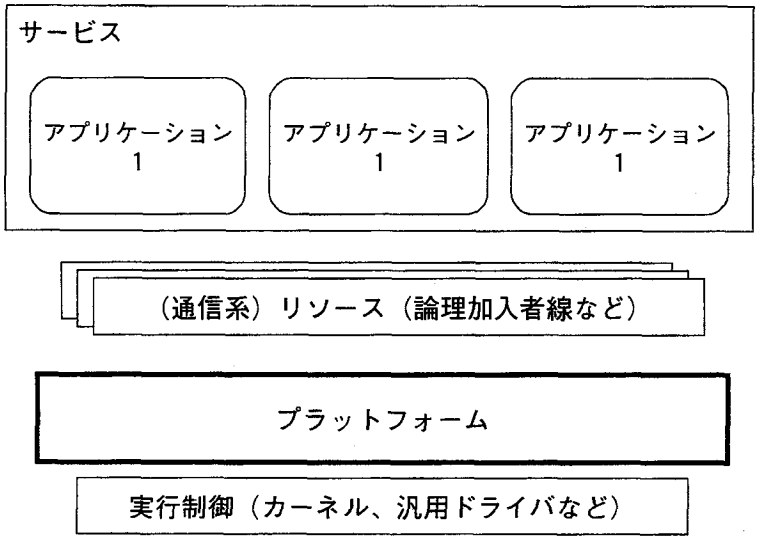


図 44 階層化構成

しかし、通信系ハードウェアがプロダクト毎に体系化されていないとプラットフォームを構築するソフトウェアが複数存在することになりプラットフォームの維持管理が複雑になる。そこでハードウェアプラットフォームを基盤として、その上に階層化ソフトウェアを構築する事が必要である。各プロダクト毎にハードウェアの基本的なアーキテクチャを統一することで、各プロダクト間で共通な装置と個別装置とにハードウェアコンポーネントを分け、それにあわせてソフトウェアプラットフォーム（以後、プラットフォームと呼ぶ）もコア部とオプション部に分けて構築する。アプリケーション毎に、オプションを選択する事になるが、コア部は必須となる。この様な考え方に従ってプラットフォームを構築するのが得策だと考える。

実行制御は、カーネルが主な機能になる。制御系資源の管理を行う。ここでは、汎用プロセッサを前提に考えている。プロセッサ系の発展を巧く取り入れていくために運用中の通信システムのプロセッサ置換が必要になる。ソフトウェアの寿命に比べプロセッサは日進月歩の進歩を遂げている。その恩恵を受けられるようなソフトウェア構成や運用方式を明確にして、初めて汎用プロセッサを使うメリットがある。多種

類のプロセッサを適用する事になるので実行制御階層でプロセッサの違いを吸収し、上位のソフトウェアに極力影響を与えないようにすることが大切である。

本章のプラットフォーム構築に関して実行制御階層は、参考文献【31】に示すカーネルを適用した。

4.2.2 オブジェクトモデル

本論文のこれまでの議論から以下のオブジェクトでプラットフォームが形成されると考える。

- コア部（共通部）
  - 再開管理
  - プラグイン制御
  - （保守）コマンド・メッセージ
  - MO(Managed Object)翻訳
  - TMN(Telecommunication Management Networks)制御
  - 所データ管理
  - 共通線制御
  - 共通部装置管理
  - ファイル更新管理
  - ファイル操作制御
  - ファイル転送
  - ローカル OpS（オペレーション・システム）
  - 運用ファイル管理
- 個別部
  - 回線交換系個別装置管理
  - ATM 系個別装置管理
  - STM 系個別装置管理
  - IN 系個別装置管理
  - ・・・・・・・・・・・

上記オブジェクトは、NOSES プラットフォームのオブジェクトモデル（**図 45**）で示す D3 レベルに相当する。どんなクラス群をプラットフォームで実現するかは、システムモデルで規定された機能条件で、何がシステム間で共通に実現できるか、何を共通に実現すべきかを熟慮して決められる。プラットフォームは、システム制御系の機能が基本になる。

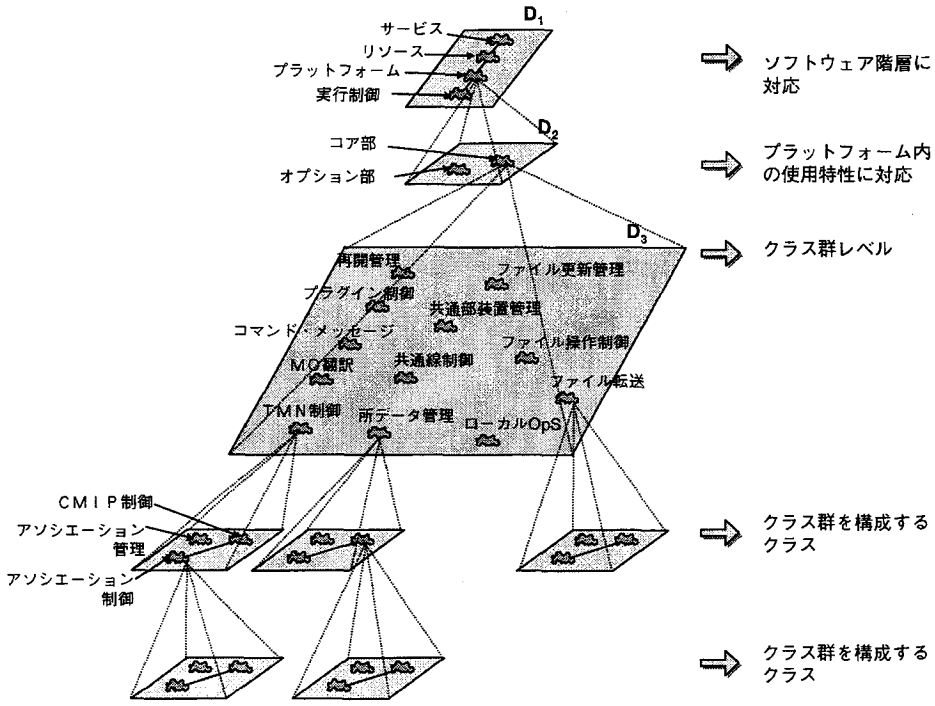


図 45 プラットフォームのオブジェクトモデル

システムの立ち上げと、何らかの異常の為にシステムがダウンした時の呼の救済処理を司るオブジェクト（再開管理）、システムの運用中にファイルのバグ対処やサービス・機能追加をユーザに影響を与えることなく実現するプラグイン・アウト制御を司るオブジェクト（プラグイン制御）、オペレーション・システム（OpS）との通信を可能にする３つのオブジェクト（(保守）コマンド・メッセージ、MO 翻訳、TMN 制御）、システムの建設時や運用中に所データの追加・変更を可能にするオブジェクト（所データ管理）、共通線制御を司るオブジェクト（共通線制御）、各種通話路装置の管理及びドライバとのインタフェースを司るオブジェクト（共通装置管理）、サービスや機能を組み込んだ新ファイルで立ち上がるための制御を司るオブジェクト（ファイル更新管理）、ディスク上で管理されるディレクトリを管理制御するオブジェクト（ファイル操作制御）、OpS から遠隔転送されるファイル制御を司るオブジェクト（ファイル転送）、ローカルに設置される保守端末を制御するオブジェクト（ローカル OpS）、各種バックアップファイルの作成を司るオブジェクト（運用ファイル管理）が必須となるオブジェクトである。他にプロダクト個別で選択するオブジェクトをプラットフォームが提供する。

4.3 カスタマイズ化

プラットフォームのユーザは、コア部と必要なオプション部を選択してプラットフォームを構築する。このプラットフォームとリソース階層の製品とサービス階層の製品を形成するクラス群からなるファイルとを合わせてターゲット用のファイルを構築する。

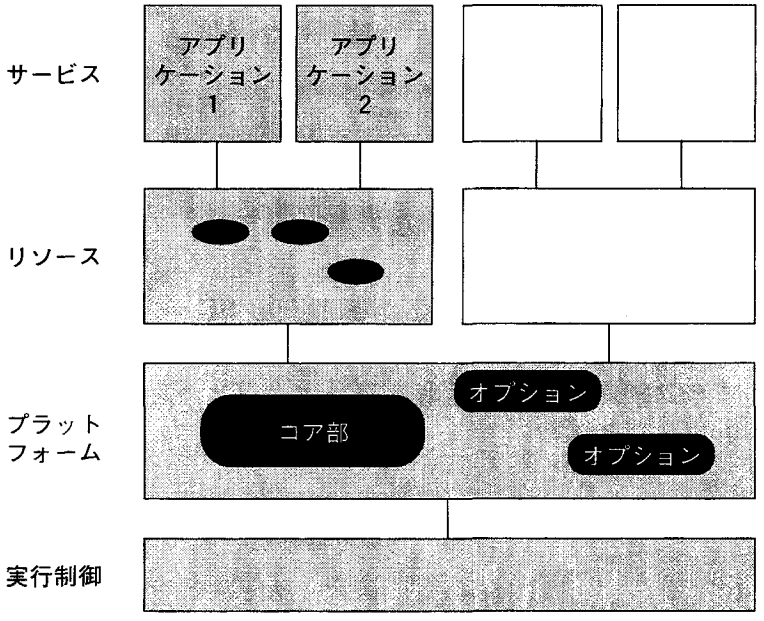
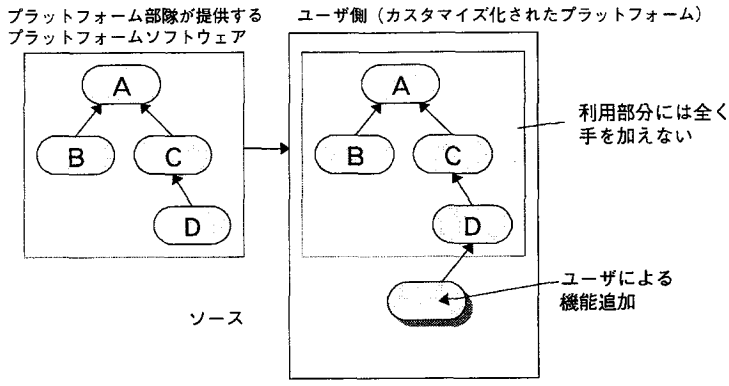


図 46 ターゲト用ファイルの構築

ユーザ側でプラットフォームのカスタマイズ化が安全に出来る事が重要である。これにはオブジェクト指向言語の機能継承が有効である（**図 47**）。言語機能を活用したプラットフォーム全体像から見た複数ユーザによるカスタマイズ化の手法を**図 48**は示す。

プラットフォームソフトウェアにユーザが機能追加をする場合、C++の継承機能を用い、既存ソフトウェアには手を加えずに機能追加（カスタマイズ化）が可能である。さらに、カスタマイズ化支援ツールにより、ユーザがカスタマイズ化してよい部分だけしかエディットできないようなガードを行うことで、安全性を高めるなどの工夫が必要である。



C++のクラス継承機能により、安全な機能追加が可能になる。

図 47 クラス継承機能を利用したプラットフォームのカスタマイズ化

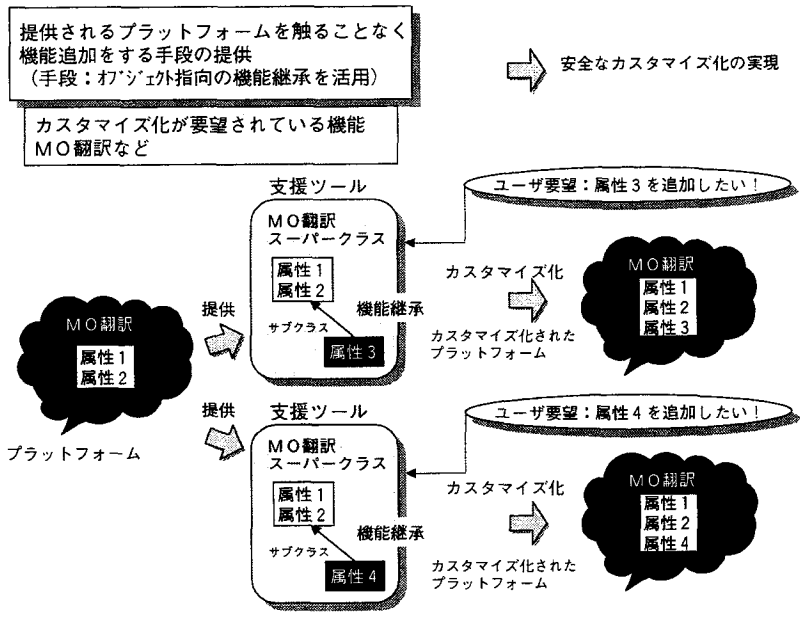


図 48 プラットフォームのカスタマイズ化イメージ

プラットフォーム部隊が提供したプラットフォームにユーザ A が、属性3の機能を持ったサブクラスを追加してカスタマイズ化しても，ユーザBは、それとは全く独立して属性4の機能を持ったサブクラスを追加してカスタマイズ化が可能になる。

C++による記述例を**図 49**に示す。スーパクラスとサブクラスの関係を回線交換用と IN(Intelligent Network) 用にカスタマイズ化する例を紹介している。回線交換側は、スーパクラスと回線交換用サブクラスを合わせて必要なファイルを構築する。IN側は、スーパクラスと IN 用サブクラスでファイルを構築すればよい。お互いに相手がどんなサブクラスを作っているかを全く意識する必要はなく独立にプラットフォームの構築を進めることが出来る。これにより通信ソフトウェア構築の効率化と技術者の持つべき知識の局所化が図れる。

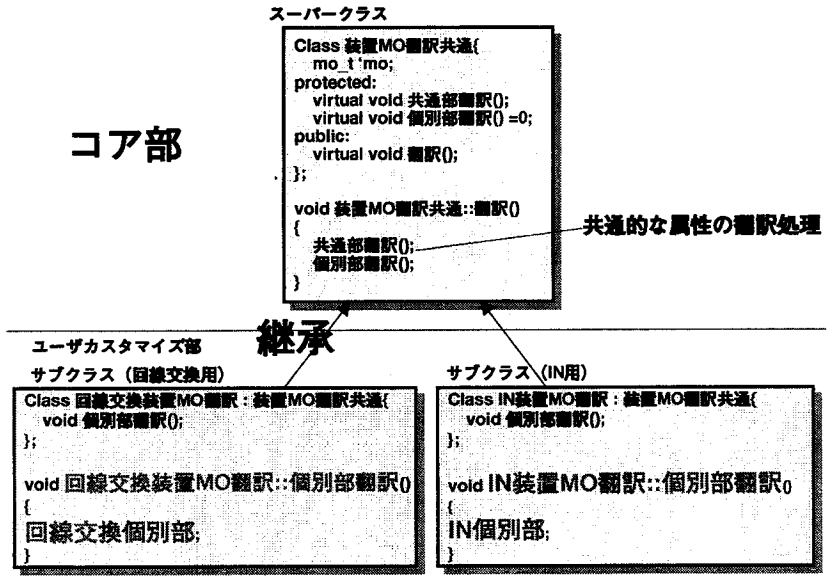


図 49　C++によるカズタマイズ化

サブクラスで差分だけ記述すればすむので、記述量の削減と試験項目数の削減に大変有効である。プラットフォームのカスタマイズ化について C++のオブジェクト指向言語が持つ特徴を活用した例を述べたが、その他にプラットフォームの利用者の利便性を考えて各種の環境を提供することも必要である。その他のカスタマイズ化支

援ツールを構築するにあたっては、プラットフォームを構成するオブジェクトが提供するサービスやそれを実現する機能によって具体的なツールの構築が異なる。

プラグイン制御などは、基本的に通信システムのハードウェアアーキテクチャに関わらないためにオブジェクトモジュール（OM）で提供できるが、再開管理は、ハードモジュール・システム構成あるいはユーザに見せるサービスの違いにより再開論理や初期設定順序が異なる事がある。この違いを再開管理が持っているカスタマイズデータを設定する事により実現できるような仕掛けが必要になる。所データ管理については、所データを格納するメモリサイズがシステム規模により異なる。この違いをカスタマイズデータで設定する必要がある。

4.4 分散制御プラットフォーム

大規模通信システムを構築するために様々な研究が行われている。本研究では、高信頼性を狙いとしたビルディング構成による大規模通信システム（**図 50**）を前提に通信ソフトウェアの研究を進めてきた。

POTS, ISDN, ATM, IN サービスや IP サービスを提供する様々なスイッチングモジュール（**図 50**では Switching Subsystem, SS）が、それらサブシステム間通信を司る ISC により結合された構成を考える。

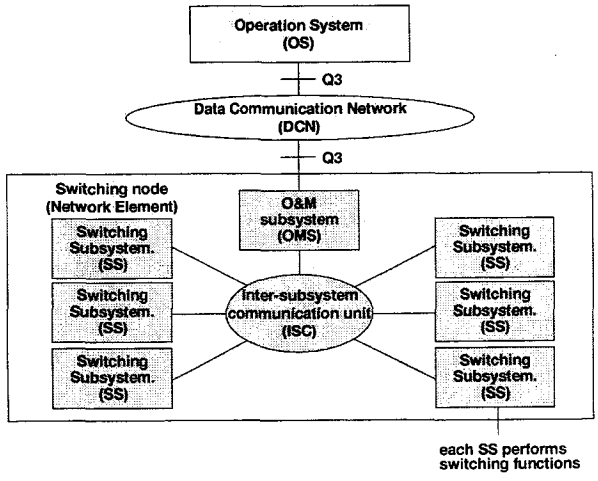


図 50　システム構成例

4.4.1 オブジェクトモデル

多様化するサービスをサブシステムの組み合わせにより実現するビルディング型システム構成では、夫々のシステム間に跨ったサービスの実現及びシステム管理の実現法が通信ソフトウェア構築の観点から重要である。分散化されたシステム内の機能の協調によりサービスを実現しユーザに提供するためにプログラムが、その機能の物理的な位置を意識していたのでは、サービスやそれを実現するソフトウェアの汎用性を損なうと共に、それらの維持管理を複雑にする。

この様な分散環境に関る問題に対に対して、本論文では言語レベルによる位置透過性と障害透過性の実現法について紹介する。

本章の前半で NOSES 機構実現に向けたプラットフォーム構築技術を提案した。後半では、プラットフォームを分散システムに適用するに当たって必要となる位置透過性と障害透過性の言語レベルによる実現を述べる。

分散制御の観点から見たサービス階層の通信モデルを**図 51**に示す。プラットフォームのユーザであるサービス階層あるいはリソース階層に存在するオブジェクト間の通信は、互いに相手オブジェクトの位置、物理的な場所を全く意識しない環境を構築することが望ましい。

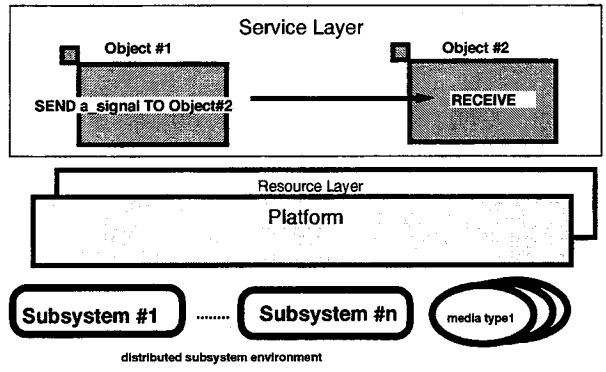


図 51　サービス階層から見た論理モデル

Object#1は、通信相手のObject#2が何処に存在するかに関らず、相手のオブジェクト名さえ知っていればSEND a_signal To Object#2と記述するだけで信号a_signalがObject#2に送られる。

この様な環境を構築する方法として、分散OSをプラットフォームが提供する方法と言語が提供する方法がある。前者の方法が現在広く提案され実現されている方法である。しかし、この分散ＯＳによる実現では、アプリケーションを記述するソフトウェアはシステムが提供するOS環境に左右されるため、今後重要と考えられるマルチベンダ環境では、その適用に限界がある。更に、プログラマはOSが提供するシステムコールを意識することになり、本論文が提案するオブジェクト間通信モデル（**図 51**）をプログラムレベルで満足することができない。

後者の言語レベルによる実現法は、本研究で提案するオブジェクト間通信モデルをそのまま言語モデルに投影する方法である。プラットフォームを利用するアプリケーションを実現するソフトウェアは、その通信モデルのみ意識した記述をすれば言語が、夫々のシステムが提供するOS環境に合わした形式でファイルを構築可能となる。

従って、本研究ではオブジェクト間通信モデルを言語モデルで実現するプラットフォームを構築し評価した。

更に、分散システムを実現するためには、上記オブジェクト間通信モデルに加えて、それら通信の土台となるサブシステム間のシステム構成管理（**図 52**）をプラットフォームで隠蔽することが必要になる。オブジェクト間通信の実現にあたり、サブシステム間通信システムが正常な場合のみ、分散不可視を実現できたとしてもアプリケーションソフトウェア構築の容易化・生産性向上を実現することはできない。即ち、オブジェクトの位置透過性の実現に加えて障害透過性の実現が重要である。

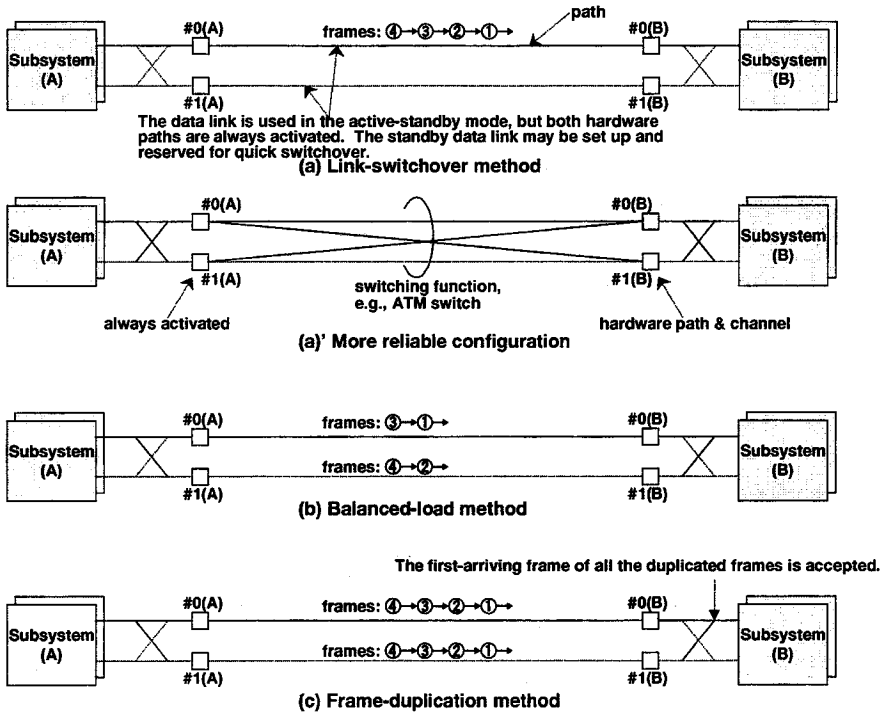


図 52　システム構成制御方式

4.4.2 言語モデル

オブジェクト間通信モデルを言語モデルに投射するには、言語コンパイラとコンパイラが出力するシステムコールとシステムの OS が提供するシステムコールを仲立ちするランライムルーチン（RTR）の提供が必要である（**図 53**）。これにより、システムが提供する OS 環境に依存しない分散環境が実現できる。

即ち、RTR は言語が提供するオブジェクトと OS が提供するプロセッサリソースなどの制御系リソースを提供するタスク環境との対応を付ける。

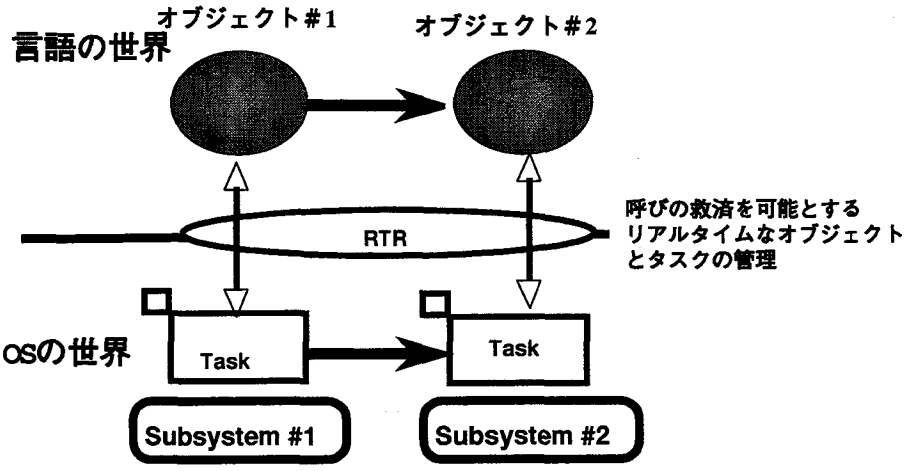


図 53　RTR による実行制御

一方、この様な分散環境をタスキング機能で持たない C や C++でも適用できる事を前提にした CORBA【26】(**図 54**) が提案されている。オブジェクト間通信を IDL(Interface Description Language)により記述し、その IDL をコンパイルすることにより OS が提供する環境とリンクする方式である。この方式は、あくまでもオブジェクト間通信に関るプロトコルを記述する言語が提供されるだけであり、アプリケーションが動作する環境は OS が提供するタスク環境をそのまま利用するものである。即ち、本研究で提案する言語レベルの分散環境の一部を擬似しているにすぎない。

従って、CORB のランタイムモデルに従った通信モデルの透過性は達成できても本研究が狙う障害透過性を同時に達成することは困難である。

言語が適用するタスキングとシステムの OS が提供するタスクとの関係を**図 55** に示す。RTR が言語の世界のリソース管理である PCB と実世界のリソース管理 TCB の関係を管理する。これにより位置透過性と障害透過性を達成する。

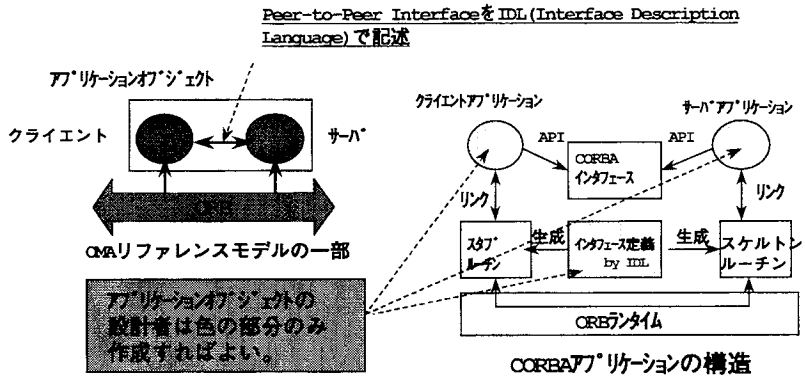


図 54 CORBA での実現方式

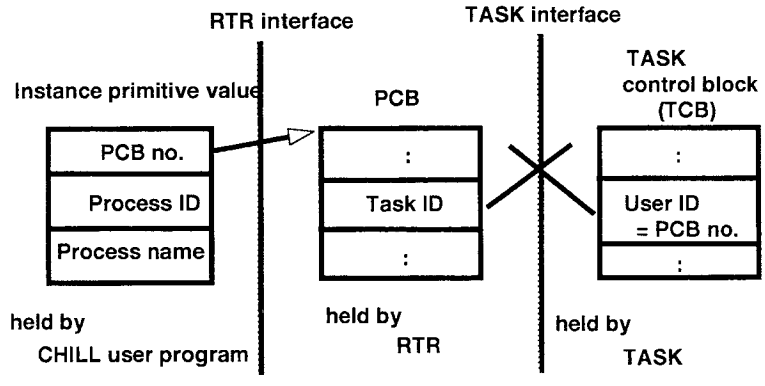


図 55　言語が提供するタスキングと OS が提供するタスクとの関係

本研究では、言語レベルで分散環境を実現する言語を持たないので、タスキング機能を持つ CHILL 言語【8】を拡張して実現し評価を行った。

オブジェクト通信モデル（**図 51**）で表現されるオブジェクトを CHILL のプロセスに対応させオブジェクト間通信をプロセス間通信として実現している（**図 56**）。従って、プロセス間通信を司る CHILL の RTR に分散環境を構築している。CHILL で記述されたプロセス間通信が CHILL コンパイラにより RTR が提供する分散通信モデルに展開される。そして異なるサブシステムに存在する RTR 間でシステム間通信が提供される。この時、RTR 内の通信自律分散制御がシステム間の通信経路（**図 52**）を管理している。RTR 間通信で通信路の何らかの異常に対して、異なるサブシステム内に存在する通信自律分散制御同士で自律的に正常な通信路を選択する。

以上の方式により言語レベルの分散制御により位置透過性を実現する。

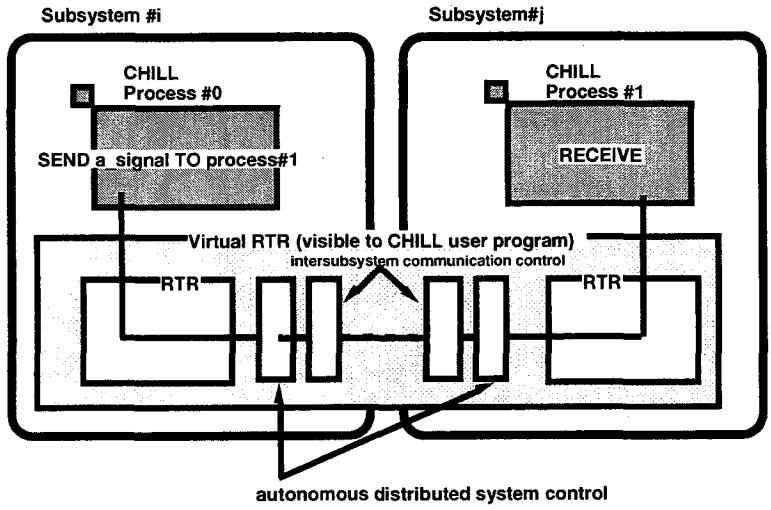


図 56　分散環境におけるメッセージ通信

4.4.3 障害透過性と呼救済

システム内の障害は、物理的な制御系リソース管理を司るタスクの異常処理として見える（**図 57**）。従来の方式では、システムの異常をタスク毎の障害処理として実現してきた。しかし、本研究で提案する言語レベルによる分散環境の世界では、プラットフォームのユーザであるアプリケーションソフトウェアレベルでは、このタスクが見えない。アプリケーションレベルではオブジェクト（本実験システムでは CHILL プロセス）で抽象化された世界しか見る事ができない。

RTR が CHILL プロセスとタスクの対応を管理する。実世界のタスクが何らかの障害処理、例えば、サブシステム間のある通信経路の障害で新たな回線が捕捉された場合、それに対応したタスクの切り替えが行われる。或いは、システム障害で、システム再開処理が実施され、ある特定の呼が救済された場合、その救済された呼びに対応した新たなタスクが生成される。しかし、サービスを提供するロジカルな世界では、その呼びに対応したオブジェクトは存在し続けている。従って、救済された呼びに対応するプロセスと新たに生成されたタスク間のリンクが速やかに形成されなければならない。この管理制御を RTR が実現する（**図 58**）。

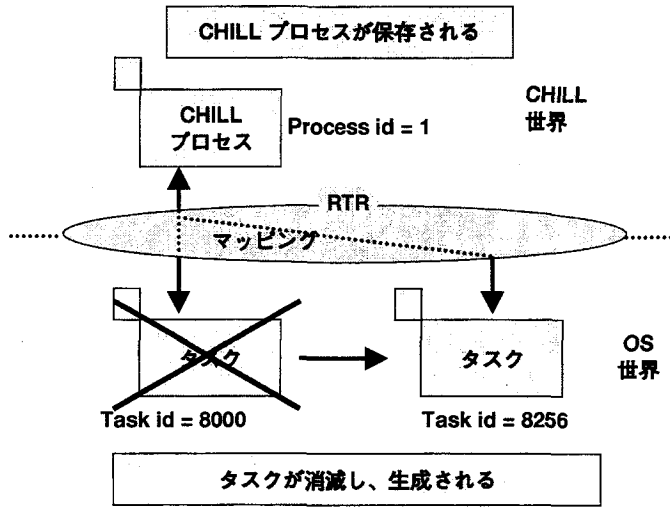


図 57 障害透過性の実現

この様な機構によりプラットフォームのユーザは、抽象的なサービスのみ意識してソフトウェアを構築すれば済むことになる。

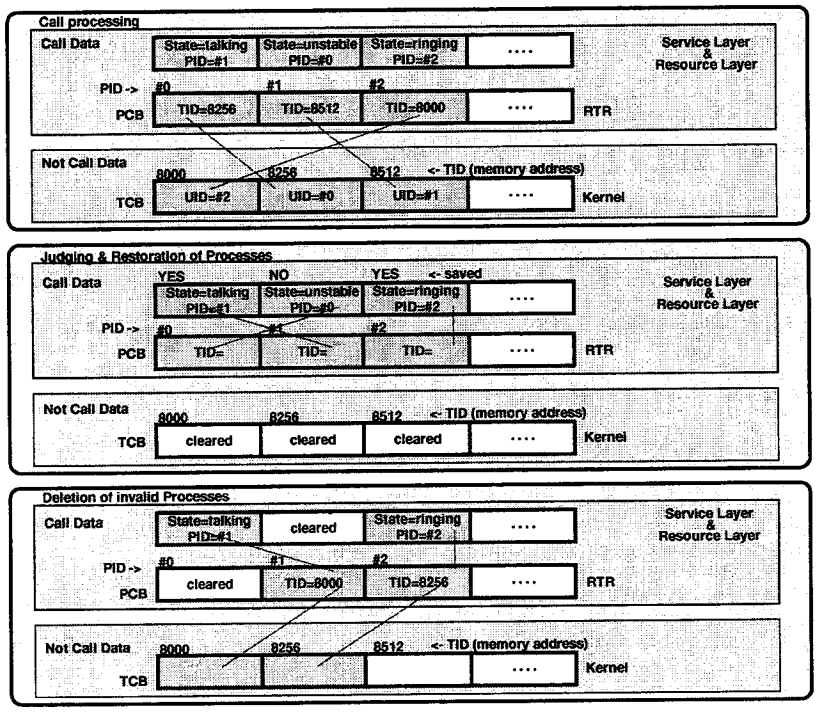


図 58　分散環境での呼救済手法

4.5 評価

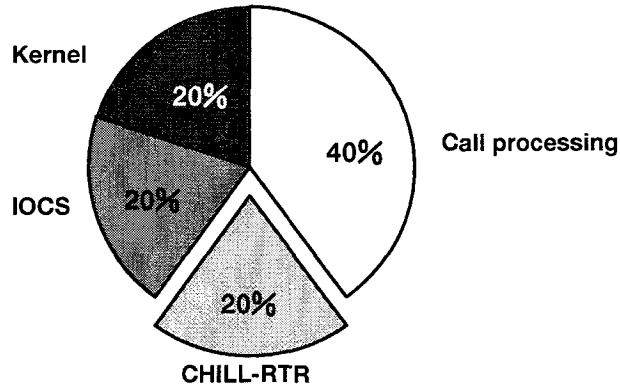
プラグイン・アウト機能のプラットフォーム化については，まず所内試験で各種サービスを提供する通信システムに適用し，その実現性が得られたことから，1996 年 12 月に導入された PHS サービス向け大容量交換システム，1997 年導入予定の大容量 ISDN 交換システム及び ATM 交換システムなどの各種通信システムへの適用が実施されている。

これにより本プラットフォームを利用するユーザは、第 3 章で提案したプラグイン・アウトを活用することができる。NOSES 制御機構のプラットフォーム化が可能になったことによる通信システムのサービス即応化の向上が大きいと言える。

言語レベルの分散不可視技術の適用については，所内実験で評価され、20%の処理増（**図 59**）がある事が明確になった。全体の 20%の処理増は、単一システムへの適用を考えた場合、経済性の問題から商用システム化が困難と判断した。その為、所内実験限りとなった。

分散環境による性能低下

ISDN基本呼処理にしめるCHILL-RTRの処理ステップ数



障害透過性による再立ち上げ時間の短縮　　**4000-erlangのシステム**

(a)ソフトウェア初期設定時間	数秒（障害透過性により短縮）
(b)ハード初期設定時間	10秒程度～（システムによる）
サービス中断時間（(a)+(b)）	10数秒～

図 59　性能評価

また、本技術の適用により全く異なる交換システムとのサービスプログラムの流通実験を行い良好な結果を得た。言語レベルでシステム依存機能を完全に隠蔽できた為，NTT の所内実験システムと全く異なるアーキテクチャを持つ海外ベンダの交換システム上で動作するサービス階層プログラムの流通性を確認できた。

ただし，システム制御に関しては，特に呼救済のコンセプトが両システム間で異なっているために評価できていない。つまり，呼救済がないシステム再開のみをサポートし，交換特有の高信頼性を追求した呼救済に関わるシステム制御の評価は対象外としている。交換レベルのサービスプログラムの流通を実現するためには，交換シ

ステムに異状が生じた場合に提供中のサービスをどのように扱うかの統一したコンセプトの確立が先ず必要である。

4.6 結言

第3章で提案したプラグイン・アウト機能を多種多様な通信システムに適用するためのプラットフォーム化技術を明確にした。本論文で提案するプラットフォームを利用するシステムは第 2 章で提案しているプラグイン・アウト機能を活用することが可能になる。

更に，システムの分散化をプラットフォームのユーザであるアプリケーションソフトウェア設計者に意識させない言語レベルの分散 OS 技術についても提案した。言語モデルが提供するタスキング機能をシステム分散制御にまで発展させることにより，アプリケーションソフトウェア設計者は，OS 環境が提供するタスク間通信を司る各種システムコールを意識することなくアプリケーションソフトウェアを設計出来る事を明確にした。

言語レベルの分散不可視技術の適用によりアーキテクチャが異なる交換システム間でサービスプログラムの流通が可能で有ることを確認することができた。

第5章システム及びネットワーク管理技術

【関連学術論文【A4】【A5】【A3】】

5.1 緒言

本章では，高信頼通信システム構築を狙いとしたビルディングブロック型分散システムへの TMN(Telecommunication Management Networks)【23】【24】適用についてコマンド応答時間の観点から評価し、分散システム構築時のプロトコルプロファイルを明らかにした。

5.2 ネットワーク管理制御

TMN は、通信装置、伝送装置、交換機等のコミュニケーションネットワークの構成要素を管理する網である。TMN の概念は、ITU-T で国際標準化が進められている。この国際標準は、TMN の形態、通信プロトコル、コマンドと自律メッセージの作り方等の OpS とノードの間の様々な取り決めがなされている。例えば、TMN では、OpS と管理されるノード（交換機、伝送装置、通信装置等）の間で送受される操作情報と通知情報を送るために、CMIP(Common Management Information Protocol)【25】を規定している。

TMN では、通信方法（通信プロトコル）に関したインタフェースだけではなく、通信プロトコルを使用したコマンドと自律メッセージを規定している。本論文ではコマンドと自律メッセージと表現しているが、TMN では、コマンドと云う言葉を使用していない。その代わりに、OpS からの操作を表すメッセージとノードからの通知を表すメッセージがある。以下では、このメッセージを自律メッセージと区別するために CMIP 形式メッセージ と呼ぶ。ここで、操作用の CMIP 形式メッセージがコマンドに、通知用の CMIP 形式メッセージが自律メッセージに当たる。CMIP では次に示すメッセージが定義してある。コマンドに対応する操作用の CMIP 形式メッセージは、5 種類、自律メッセージに対応する通知用の CMIP 形式メッセージは、1 種類規定されている。

CMIP で規定されたメッセージ

- Create ： MO(Managed Object)インスタンスの生成操作
- Delete ： MO インスタンスの消去操作
- Set ： MO インスタンスの属性値の設定操作
- Get ： MO インスタンスの属性値の読みだし操作
- Action ： MO クラスで定義された動作の実行指示操作
- Event_report ： MO インスタンスからの OpS に対する通知

MO は、管理される対象であるもの、例えば通信システム内の装置等を示す。MO インスタンスは具体的な管理される対象、例えば通信システム内の 11 番目の加入者装置等を示す。OpS から見ると通信システムは、MO インスタンスの固まりに見える。OpS では、この MO インスタンスに対して、メッセージを飛ばして様々な管理操作を実現することになる。Create と Delete は、MO インスタンスの生成と消滅を指示する操作である。Action は、MO インスタンスに動作開始を指示する操作を示す。動作として、例えば、プラグイン、プラグアウト、系構成の切り替えなどがある。

通信システムは、MO インスタンスから警報が発生したことや、MO インスタンスの状態が変化したこと等を OpS に対して、Event_report で通知する。

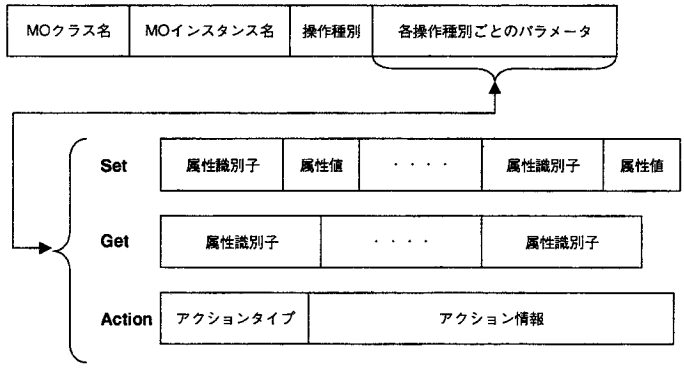
MO は、属性を持っている。属性は、属性識別子と属性値からなる。これは、MO に関する情報を蓄えるデータベースに相当する。データベースのある項目の見出しが属性識別子に、データの内容が属性値に相当する。このデータベースには、OpS から見るあらゆる情報が蓄積されている。MO に蓄積される情報は、MO の名前や作成メーカ等の MO 自身の管理用の情報、使用中状態や未使用状態に当たる状態を表す情報、しきい値や起動時刻等の制御関連のパラメータ情報など多種の情報からなる。これらの情報には、すべて属性識別子という名前がふられている。Get では、この MO インスタンス名と情報が知りたい属性の属性識別子を指定する。OpS から通信システムに対して Get が送られると、通信システム側からその Get の応答が返される。OpS は、この応答により、MO インスタンスの情報として属性値を得ることができる。

MO インスタンスは、属性、動作、通知等の性質を持っている。ここで、同一の属性や動作を持つ MO インスタンスを集めてくる。この集合体 を規定するのが MO クラス である。例えば、装置を表す MO クラスや、ソフトエアに対応した MO クラスがある。装置に対応する MO クラスは、属性として使用中、未使用等の状態にあたる情報、作成メーカ名、障害発生状況の情報等の属性を持つ。動作としては、状態を変更するコマンドに相当するものを持つ。ソフトエアに対応した MO クラスでは、装置クラスとは異なる属性や動作が規定される。**図 60**に CMIP 形式メッセージの概要を示す。また、**図 61**に TMN でのオペレーションイメージを示しす。

CMIP 形式メッセージに乗せられる MO クラス、MO インスタンス、オペレーションタイプ というパラメータは、各オペレーションに共通である。OpS では、これらの共通な情報の他に、操作種別に従った情報をノード側に送る。Set では、セットすべき属性をあらわす属性識別子とその値である属性値のペアを 1 個以上指定して送る。

Get では、属性識別子のリストが送られ、指定された属性の属性値が応答として返される。Action では、この動作の種類を識別すべき情報であるアクションタイプと、より詳細な動作に関するパラメータを指定するアクション情報を送る。

図 61オペレーションのイメージを示している。図中のノード側の操作を受ける側という意味でエージェント側と呼ぶ。OpS 側をマネージャ側と呼ぶ。マネージャ側からエージェント側の MO への操作が送られる。その逆に、エージェント側からマネージャ側に操作への応答と、通知とが送られる。図中左方に、MO インスタンスの特徴を表す情報を示す。MO の特徴を表すものとして、MO クラス名、MO インスタンス名、属性識別子と属性値をペアとしたリスト、受けることのできるアクションタイプ、出すことのできる通知種別等を定義する。



（注：基本的なパラメータのみを記述）

図 60　　CMIP 形式メッセージの主なパラメータ

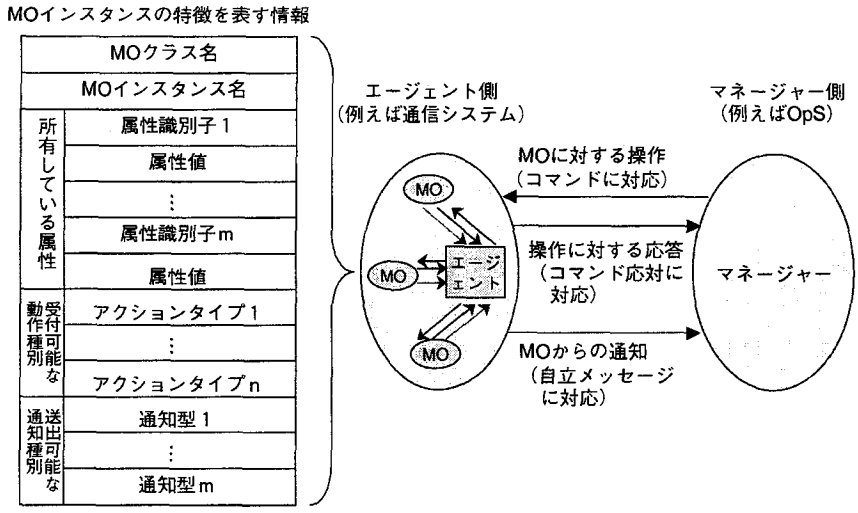


図 61　TMN でのオペレーションのイメージ

5.3 評価

高信頼通信システムの構築に当たっては、4.4 項で前提としたビルディングブロック型分散システム(**図 50**)を考える。分散システムへの TMN 適用にあたって、コマンド応答時間を最小にする TMN プロトコルを提案する。

分散システムは、各種サブシステムが通信制御を司る ISC により結合される構成を考える。

オペレーションシステムは、CMIP 形式のメッセージに相当するコマンドをシステムに送信し、自律メッセージを監視することによりシステムを管理制御する。例えば、プラグインコマンドを送信し、プラグイン完了の自律メッセージを監視する。TMN コンポーネントで重要な機能がエージェントである。このエージェントが、O&M サブシステム（OMS）とオペレーションサブシステム(OS)間の制御に関るプロトコルプロファイルを決定づける。

5.3.1 プロトコルプロファイル

マネージャとエージェントの関係に着目すると以下の３形態に分類可能である(**図 62**)。

Operation(a): OMS オペレーション

この形態は、OMS 自身のオペレーションに相当する。Ｑ３インタフェースにより OS が OMS にコマンドを送出する。エージェント機能が OMS に存在する。

Operation(ｂ): ノード全体のオペレーション

この形態は、ノード全体のマネージメントを提供する。ＯＳは、OMS に存在する OA&M ソフトウェア（ＭＦ）にコマンドを送出する。ターゲットにするＭＯは、ノード全体に存在する。ＭＦがそのコマンドを受けて、対応するサブシステム（ＳＳ）に内部コマンドを送出する。この内部コマンドは OMS が持つマネージャにより新たに生成された CMIP オペレーションによりＱｘインタフェースを通して運ばれる。

Operation(ｃ): ＳＳオペレーション

この形態は、SS 自身の管理・制御である。マネージメントデータが OS から OMS を通してＳＳへ送られる。

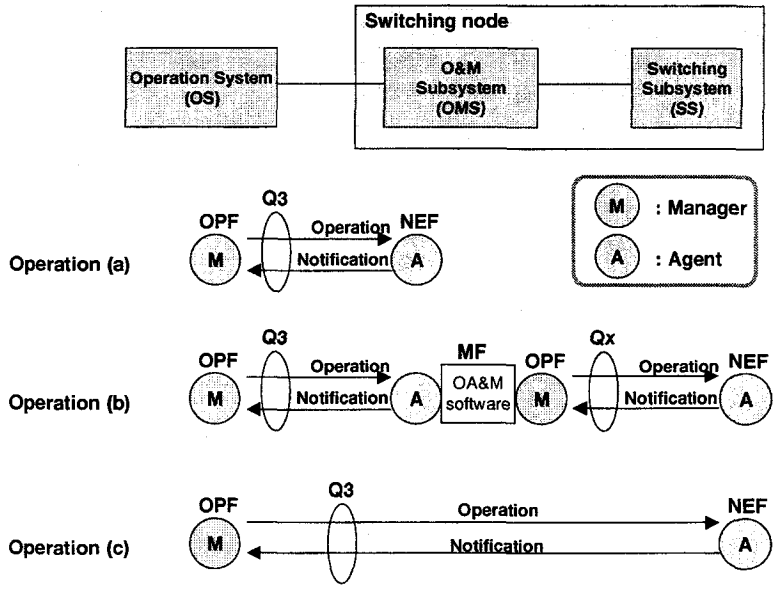


図 62 マネージャとエージェントとの関係

夫々のオペレーション形態に対して幾つかのプロトコルスタックが考えられる。効果的なデータ転送を可能とするために、**図 63**が示すマッピング機能を採用している。即ち、上位方向と下位方向の２つのマッピングにより構成する。(n-1)レイヤのプリミティブが、あるＮレイヤのプリミティブに対応する事を利用して、下位方向のマッピングをＮレイヤのプリミティブにマッチする(n-1)レイヤのプリミティブを選択することにより実現する方式である。この時、上位方向のマッピングは、(n-1)レイヤのプリミティブを使ってＮレイヤのプリミティブを特定できないので(N+1)レイヤプロトコルのヘッダに存在するキー情報を上位方向のマッピング時にサーチすることによりＮレイヤのプリミティブを得る方式である。

このマッピング機能によりレイヤ４、５、６を省略することができる。この様な機能は、CMIP を利用するアプリケーションレイヤプロトコルにとって必要な機能ではない。

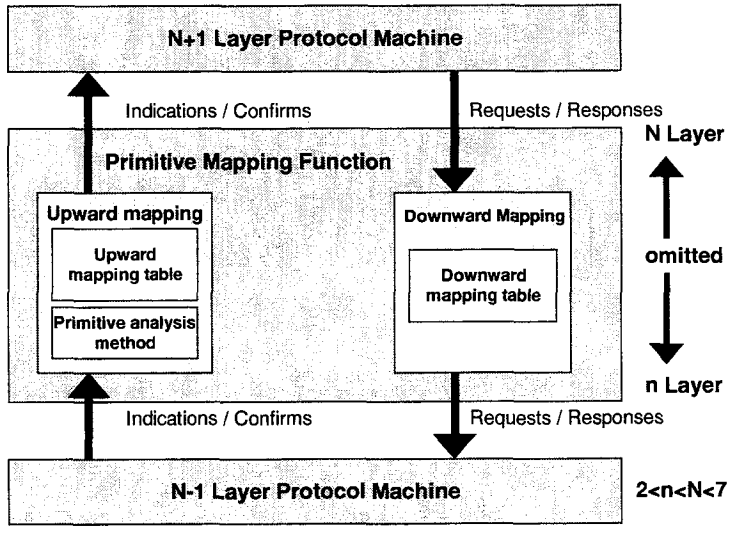


図 63　マッピング機能

OMS 自身のオペレーション形態である Operation(a)は、オペレーションコマンドの応答時間評価の観点からは分散としての特徴に関らない基本的な形態である。ま

た、応答時間評価の観点からは形態（b）が最も厳しい条件となることから形態（b）に着目する。

両形態共に、OS は Q3 インタフェースにより OMS に接続している。OS と OMS 間は、OSI の 7 レイヤプロトコルスタックを持つ。Operation(b)では、OMS が Qx インタフェースにより SS に接続される。この Qx インタフェースに着目したプロトコルプロファイルとして 2 つの形態がある（**図 64**）。

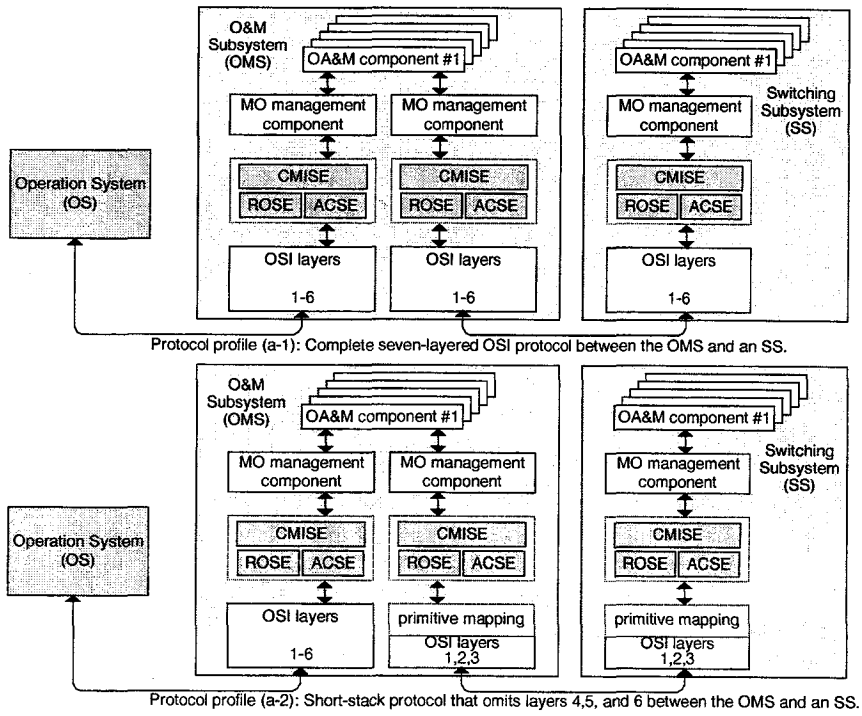


図 64　ノード全体のオペレーションプロトコル

・Protocol profile(a-1): OMS と SS 間に 7 レイヤを適用
・Protocol profile(a-2): OMS と SS 間でレイヤ 4, 5, 6 を省略する
の 2 形態がある。

Qx インタフェースとして 7 レイヤを用いる必要がないことからノード全体のオペレーション形態 Operation(b)としては、Protocol profile(a-2)を適用することが、コマンド応答時間の観点から望ましい。

SS 自身のオペレーションとしては、Q3 ｲﾝﾀﾌｪｰｽが SS で終端することになる。そのため OMS と SS 間の内部インタフェースとして次の２形態が考えられる（**図 65**）。

・内部プロトコル形態(b-1)：　レイヤ３以上をリレイする方式
・内部プロトコル形態(b-2)：　レイヤ６以上をリレイする方式

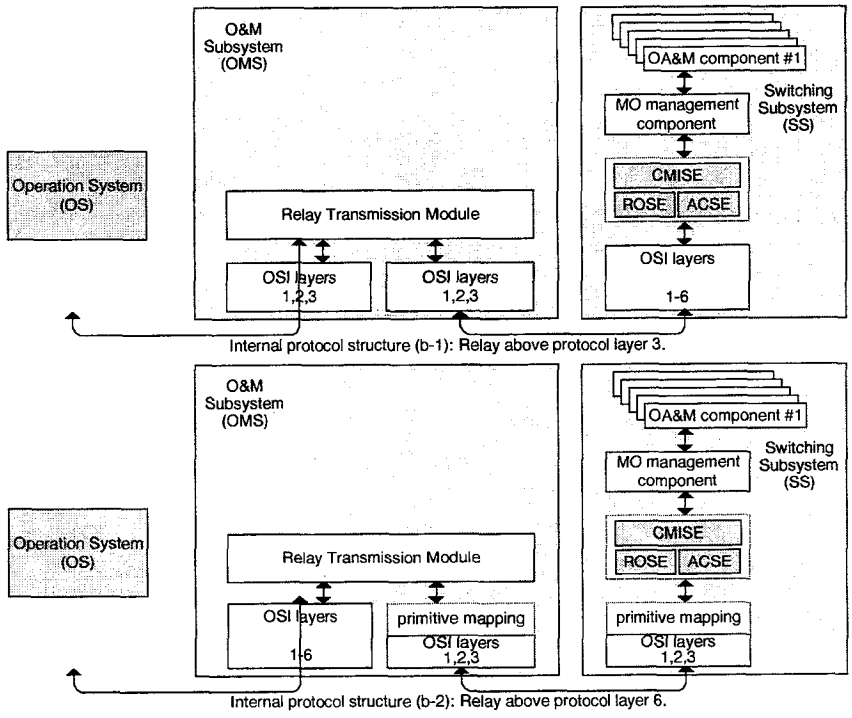


図 65　サブシステム内のオペレーションプロトコル

5.3.2 コマンド応答時間評価式と数値

コマンド応答時間の評価において、ユーザから見た心理的な許容時間が問題となる。CCITT【32】では、その許容時間を２秒として勧告している。ネットワークの転送時間を考慮すると、ノード内で許容される時間は 500ms 以下と考えられる。いかなる CPU の使用状況においても平均的なコマンド応答時間としては、この 500ms 以下を満足する必要がある。

平均コマンド応答時間を次の３つに分けて考える。

・OMS 内での遅延時間
・OMS と SS 間の転送時間

・SS内の遅延時間

である。

コマンドに対するOMS及びＳＳ間の平均遅延時間はCPU内の待ちキュー時間と処理時間により決まる。待ちキュー時間は、M/G/1 タイプの非優先キューより求めることが出来る。コマンドメッセージ転送に関るTMN処理は非優先処理をして扱われる。全てのコマンドはOMSを経由して夫々のＳＳへ送られることから、OMS内のTMN処理はSSの数に比例して増加する。

平均コマンド応答時間は、

T = WOMS x NOMS + DOMS + WSS x NSS + DSS

より得られる。

- WOMS:　OMS内の平均待ち時間
- NOMS:　OMS内での１コマンドが処理される間に待ちキューに入るユーザの数
- DOMS:　１コマンドあたりのOMS内処理時間
- WSS:　SS内のキューの平均待ち時間
- NSS:　SS内で１コマンドが処理される間に待ちキューに入るユーザの数
- DSS:　１コマンドあたりのSS内処理時間
-

平均サービス時間としては内部プロトコル形態(b-1)の場合、優先ユーザのＳＳ内平均サービス時間は0.22ms、最も低いユーザの場合は1.2ms、そして夫々のユーザの平均サービスタイムは1.2msである。OMS内の夫々のユーザの平均サービス時間は0.65msである。NOMSが2、NSSが5、DOMSが1.29ms、そしてDSSが5.89msである。内部プロトコル形態(b-2)の場合、上記に対して夫々、0.22ms, 1.6ms, 0.65msであり、NOMSが4、DOMSが2.60ms, NSSが3、そしてDSSが4.70msである。

図 66に平均コマンド応答時間の評価結果を示す。ＳＳの呼処理が85%と95%のCPU使用時間時の応答時間とSS数の関係を示している。

SSの数が増加するとOMS内のコマンドメッセージが集中する。その結果、OMS内の待ち時間が増加することになる。SSのCPU使用率の増加によりSS内の待ち時間が増加する。

内部コマンド形態(b-1)の場合、95%のCPU使用率で要求される500msのコマンド応答時間を満足出来ないが、(b-2)の形態は満足可能である。OMSが25以下のSSに接続されていれば形態(b-2)の方が形態(b-1)よりも優れている。25台以上であると、

逆転している。これはＳＳの遅延時間が、ＳＳ数の少ない時は支配項となり、SS の数が増加すると OMS の遅延時間が支配項となるためにである。

分散システム構成では、ＳＳが 20 システム以下であれば内部コマンドとしては形態(b-2)が優れている。

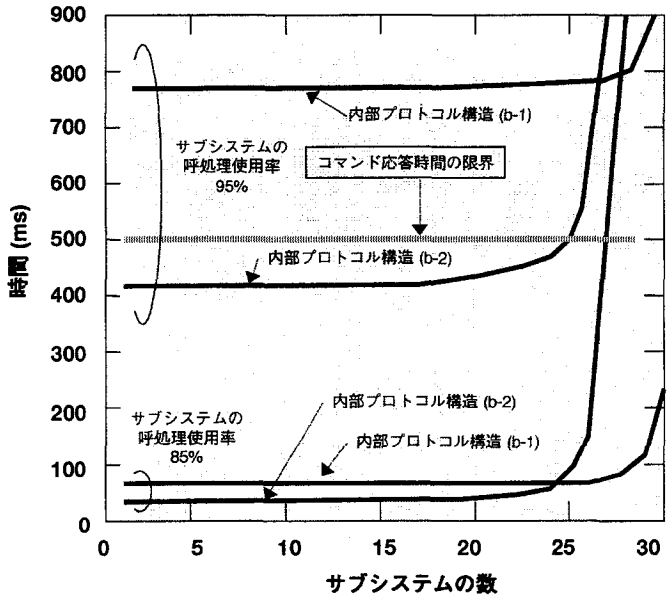


図 66　平均応答時間評価

5.4 結言

高信頼通信システム構築を狙いとしたビルディングブロック型分散システムへの TMN 適用に向けたプロトコルプロファイルを明らかにした。これにより本論文で提案するプラグイン・アウト処理に必要となる各種コマンドや自律メッセージを効率よく実現可能となる。

第6章 通信ソフトウェア開発支援技術

【関連学術論文【A3】】

6.1 緒言

本章では，本研究で提案するオブジェクト指向通信ソフトウェアの開発に向けた開発支援技術について提案する。オブジェクト単位のサービス・機能追加の着脱とプラグイン・アウト機構を実現するプラットフォームの共通化の両技術を支援する。また，物理的なシステム構成にフリーなサービスプログラムの設計支援技術を提案すると共に、支援技術構築の観点から、本論文で提案するオブジェクト指向に基づく階層化プログラム構造について評価を行う。

6.2 流通と流用

ソフトウェアの再利用は、効率よくソフトウェアを開発・維持管理するために大変有効である。しかし、この実施方法を誤るとソフトウェアを再利用したためにかえってソフトウェアの開発・維持管理が難しくなることがある。また、部品を登録しても他で再利用しない状況になりかねない。そこで、ソフトウェアの再利用で“流用”と“流通”を明確に使い分けることを提案している。大きなプロジェクトで大規模ソフトウェアの再利用を図るにはこの使い分けが必須と考える。

- ソフトウェアの流通
  - 提供側で製造したソフトウェアを無修正で使用すること。修正とは、例え1文字でもソースファイルを書き換えた場合を指す。なお、所データなどの運用データの変更はソースファイルの変更とは考えない。
- ソフトウェアの流用
  - 提供側で製造したソフトウェアを修正して使用する。

従って、SM（ソースモジュール）による提供されるソフトウェアは、修正が前提となるために流用となる。OM（オブジェクトモジュール）により提供されるソフトウェアは、コンパイル、アセンブル後のＯM形式、OM をリンクしたリロケータブル LM 形式、OM をマージしたアーカイブファイル形式のいずれかの形態で提供されるソフトウェアであり、ユーザ側でソースファイルの修正ができないので必然的に流通となる。LM（ロードモジュール）により提供されるソフトウェアは、リンケージ後のアブソリュート LM 形式であり流通となる。

プラットフォームソフトウェアの運用形態として、そのプラットフォームソフトウェアを OM 或いは LM レベルで使用する場合をここでは基本運用形態、ユーザ側で修正して使用する形態を応用運用形態と呼ぶことにする。

ソフトウェアの再利用を促進するためには、提供側も利用側も再利用可能な範囲を拡大できる配慮が必要である。つまり、同じコンセプトの下でお互いに設計を進める必要がある。本論文では以下を基本に考えている。

- OM 提供ソフトは定められた単位（OM 提供単位）毎に、流通可能とする。これによりユーザは、ユーザ独自のソフトウェアを組み合わせてシステムを構築可能となる。
- 提供ソフトウェアはワークステーション上でのデバッグ終了レベル、ターゲットシステム上で立ち上げ可能レベル、ターゲットシステム上での正常系確認レベルなどのいくつかの段階に分け、保証範囲を指定して、仮提供することにより並行開発可能にすることに配慮する。
- SM 提供ソフトは、ユーザの希望する範囲で可能な限り、容易にチューン可能な構造とする。

カスタマイズ化支援技術

運用形態を**図 67**に示す。ソフトウェアの再利用を安全にかつ効率良く表現するために再利用ツール（プラットフォームカスタマイズ化ツール）を構築している。

ソフトウェアの維持管理コストを考えると、複数種のハードウェアモジュールに搭載された場合でも、プラットフォームソフトウェアは一種類のファイルであることが望ましい。しかし、(a) ハードウェアやシステムの属性の違い（メモリ実装、プロセッサの制御形態、通話路系ハードウェア形態、など）により同一プログラムでは対応しにくい、(b)ユーザ毎の要求条件の違い（再開エスカレーション論理の違い、ファイル更新処理の違い、など）があるため、ユーザ側で機能の一部を修正して使用したい、さらに(c)システムにより必要な機能が異なる、ということから、ユーザ要望に応えるためのカスタマイズ化が必須である。

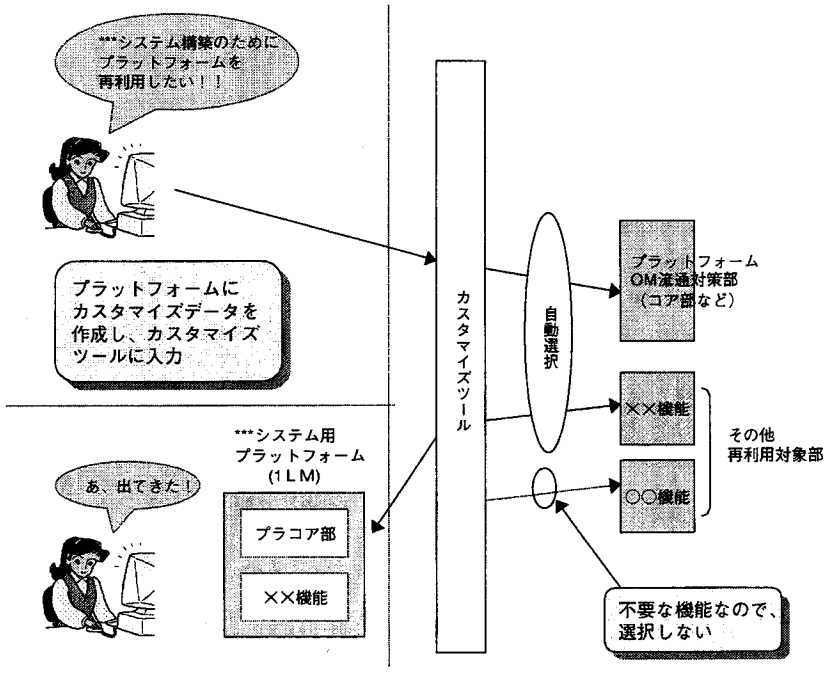


図 67　運用形態

図 67が示すカスタマイズ化ツールにより上記(a)(b)(c)の要求条件を満たす環境を以下の手法により提供する。

- ユーザは、CS データ（カスタ支援データ）を作成する。CS データには上記(a)の様なシステム構成の違い及び(c)で述べた機能の要否を記述されている。
- 処理の異なる部分（上記(b)）については、ユーザが規定されたフォーマットで差分機能を記述する。
- カスタマイズ化ツールではこれらの機能によりシステム構成に合わせたカスタマイズ化を実施する。
- 実施後、CS データで記述された必要機能に相当するファイルを集めて最終的なファイルを作成する。

この様なソフトウェアの再利用とカスタマイズ化はオブジェクト指向のクラスが持つモジュール化、抽象化、カプセル化そして機能継承機能が有効である。

6.3 部品化とライブラリ

第2章で述べた様に、オブジェクト指向通信ソフトウェア開発において**図 10**が示す部品化の手法を用いている。クラスを部品の基本単位とし、一部のライブラリを除いたすべての部品はクラスとして実現している。この部品化手法においてクラス間の継承と階層化されたプログラム構造を活用することによりサービス階層のプログラム自動生成化の可能性が広がる。呼処理サービスの形式的仕様化技術とプログラムの自動生成技術については次節で紹介する。

ライブラリとは、ヘッダファイル（宣言やマクロなど）と、コンパイルされたファイル（OM）の集合であり、ユーザは、用意されたヘッダファイルをインクルードすることでライブラリを利用することが可能である（**図 68**、**図 69**）。

ライブラリには、関数ライブラリ、マクロライブラリ、クラスライブラリの種別がある。

クラスライブラリは、クラスの定義を記述したヘッダファイルと静的データメンバの定義、メンバ関数の定義ファイルをコンパイルした OM からなる。ユーザはドキュメントやソースブラウザにより利用したいクラスを見つけると、そのクラスのヘッダファイルをインクロードする。

このクラスライブラリに、本論文で提案するプラグイン・アウト機能実現に関わる一連のクラスを予め準備しておけば、呼処理プログラム設計者は、サービスに関わるクラスのみ記述し、プラグイン・アウト関連クラスをインクルードすれば必要な NOSES 機構を持ったファイルが自動的に構築可能となる。

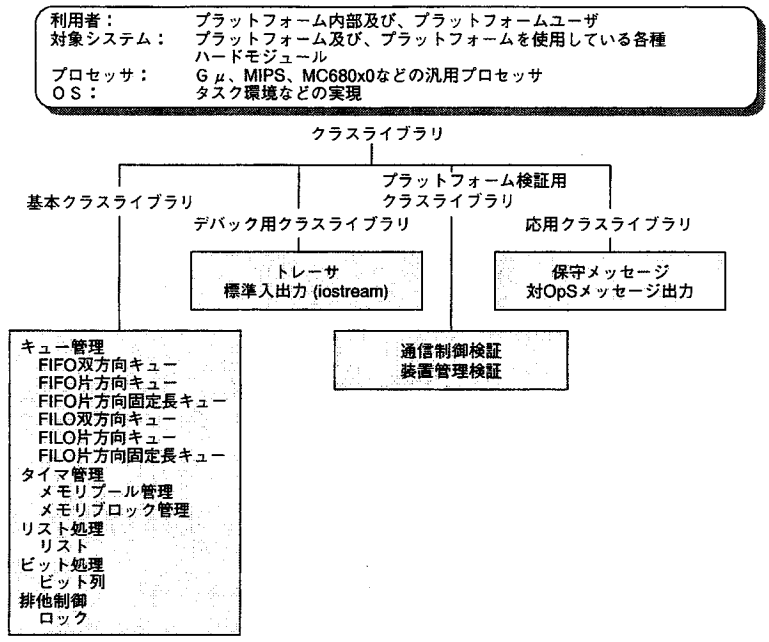


図 68　クラスライブラリ管理の全体像

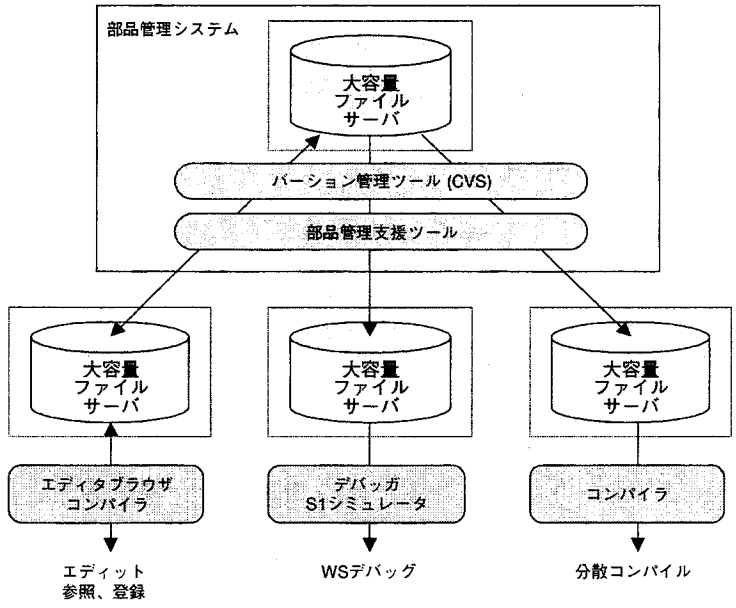


図 69　部品管理システム

6.4 評価

本論文で提案している第2章オブジェクト指向通信ソフトウェアと第4章プラットフォーム構築技術について、サービスプログラムの自動生成の可能性とプラットフォームの共通化の観点からソフトウェア開発支援技術の評価を行った。なお、後者の共通プラットフォーム化に関する評価は第4章の第5節で紹介しているので、本節では前者のサービスプログラムの自動生成について評価を述べる

6.4.1 サービスプログラムの自動生成

形式仕様記述

ソフトウェアの CAD 化を実現するためには、曖昧性のない Formal な仕様記述が重要である。汎用手法はないが、適用分野を限定して適切な記述モデルを用意するならば、簡明かつ Formal な仕様記述も可能である。ここでは、交換システムの仕様記述として CCITT で勧告化されている仕様記述 SDL／PE【33】と Prolog【34】を用いて、本論文で提案する通信ソフトウェア構造により呼処理プログラムに関する自動生成の可能性について評価する。Prolog を用いて記述することにより、形式的かつ宣言的実行可能な仕様記述が可能となる（**図 70**）。

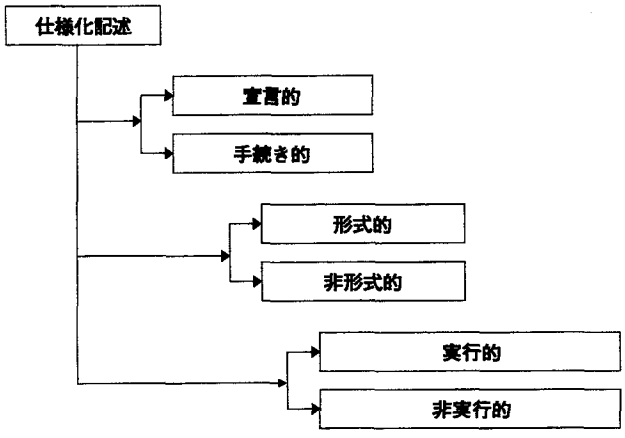


図 70　仕様化記述

抽象化された呼処理プログラム

第 2 章で呼処理プログラムは、抽象化された論理リソースを制御することでモデル化できるとを述べた。その一例を**図 71**に示す。この例は、数字受信中に第一数字を受信する場合の仕様を示している。第一数字を受信し、正常な数字を受信すれば継続する数字を受信する状態に移行するためのタスクを起動し、論理リソースとして表現されるダイヤルトーン（DT）を解放する。

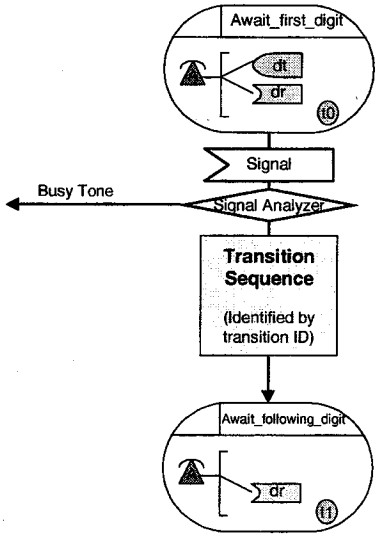


図 71 SDL/PE により表現される抽象化された呼処理

宣言的実行可能な仕様記述

交換システムの呼処理仕様を Prolog で記述する（**図 72**）。

呼処理の状態は、state(呼処理状態名、[論理リソース名 | その状態]、、、)により、分析トリーは、**図 73**によりモデル化できる。また、状態遷移は推移則が成り立つ。即ち、start→Next & Next→Target により Start→Target であるから、以下の一般則が成立する。

Resspec(Rname, Start, [Outhead|Outtail], [Inhead|Intail], Target)

```
:- resspec(Rname, Start, Outhead, Inhead, Next),
  respec(Rname, Next, Outhead, Intail, Target).
```

トリー型データにモデル化される分析処理は、

```
callerAnalyzer([a1|T], R):-b(T,R).
callerAnalyzer([a2|T], R):-c(T,R).
b([b1|_], transitionID1).
b([b2|_], transitionID2).
c([c1|_], transitionID3).
c([c2|_], transitionID4).
c([c3|_], transitionID5).
```

として記述できる。?-signalanalyzer(a2,c2,x). と質問すれば x=transitkionID4 が得られる。即ち、タスク番号が得られる。

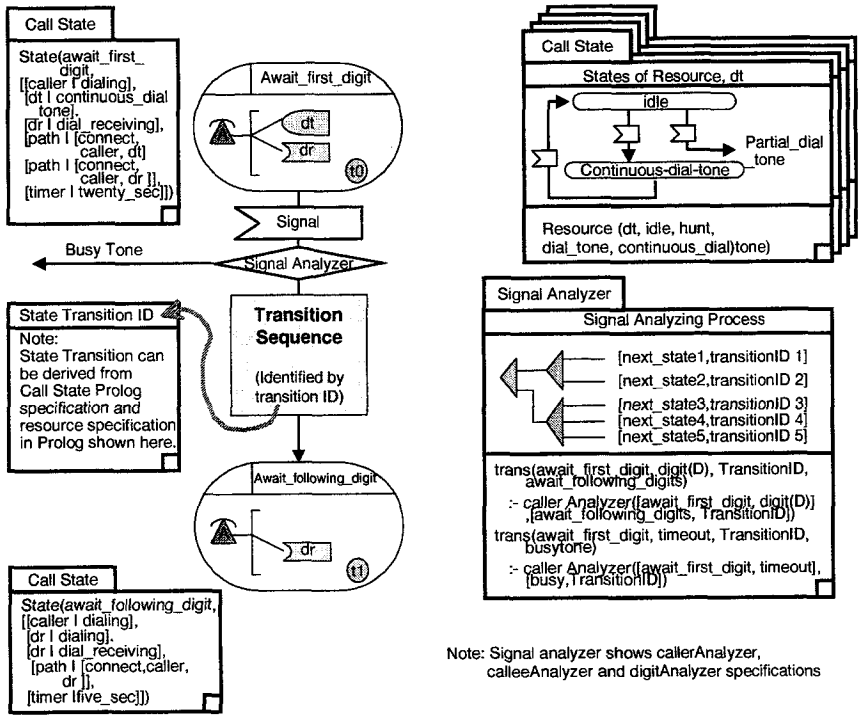


図 72　SDL/PE と実行可能な宣言的仕様記述例

呼処理の状態遷移情報 trans も、現状態、入力信号、次状態を与えることで規定できる。

Trans(現状態名、入力信号、次状態名)

:-callerAnalyzer(現状態名、入力信号、各種情報、次状態名、TrasisionID).

以上の様に SDL／PE に対応した Prolog による形式的な仕様記述が可能になる。これらを用いて状態遷移の内容（遷移時に各共通リソースに送るべきメッセージの並び)、即ち、タスクを計算機支援により求めることが可能になる。

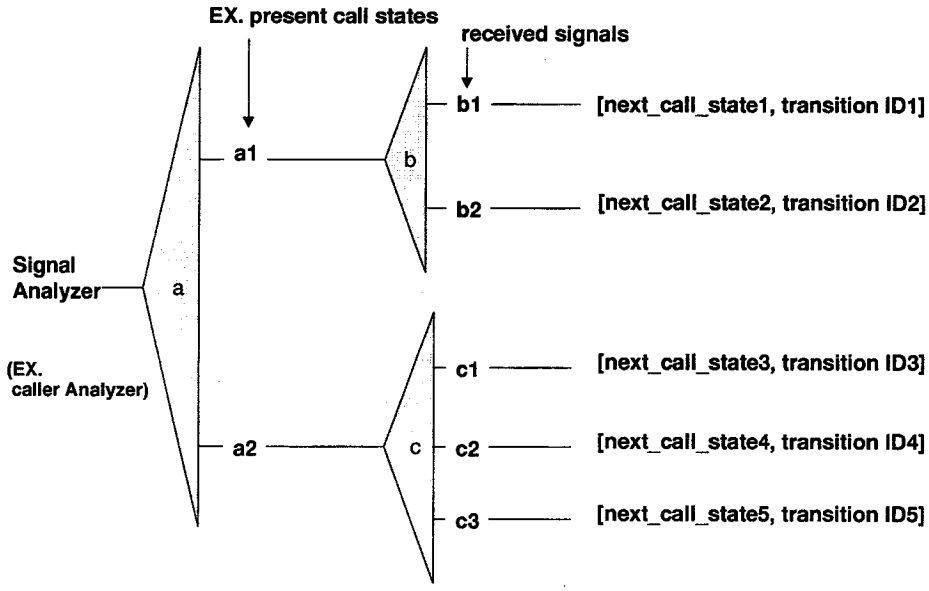


図 73　信号分析・サービス分析

形式仕様記述を図 74に示す。

計算機支援設計

サービス階層に相当する呼処理サービス仕様を Formal に記述し、さらにそのサービスが制御する論理リソース仕様を Formal に記述することにより状態遷移を司るタスクを生成可能である。従って、さらに必要とする NOSES 機構関連クラスを継承することにより、必要となるクラスが得られることになる。これで仕様記述によるプログラムの自動生成が可能となる**図 75**。

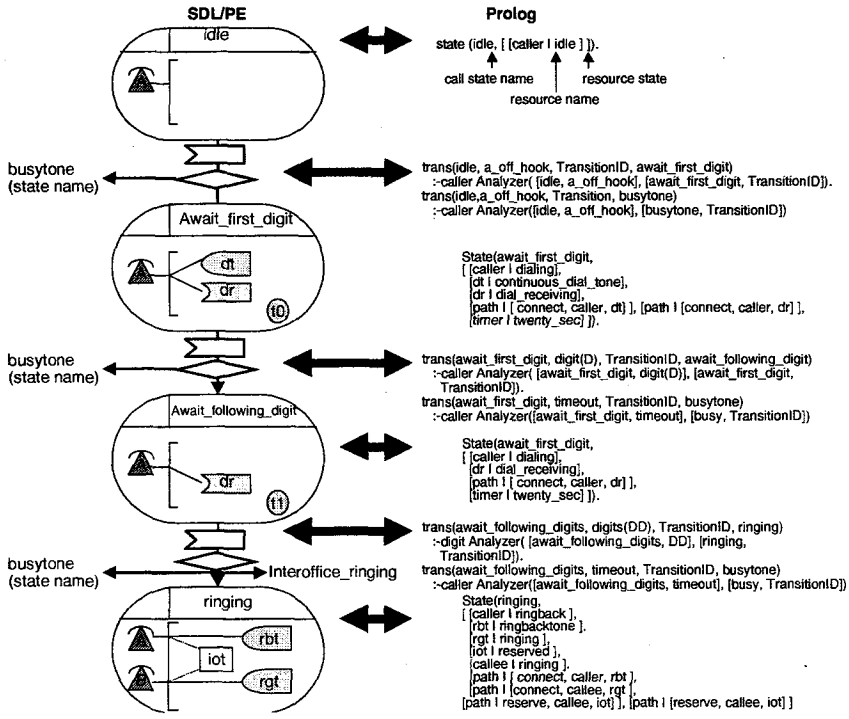


図 74　形式仕様記述

6.5 結言

本研究で提案する通信ソフトウェアの開発に向けた開発支援技術について提案した。オブジェクト単位のサービス・機能追加の着脱とプラグイン・アウト機構を実現するプラットフォームの共通化の両技術を支援する技術を提案している。第 2 章及び第 3 章で示した生産性の向上は，ここで示した，支援技術に負うところが大きい。

また，物理的なシステム構成にフリーなサービスプログラムの設計支援技術の提案と評価についても紹介した。本研究で提案している階層化構成によりサービスプ

ログラムは論理的なリソースオブジェクトを如何に処理するかに着目すればよい。その結果，本研究で提案している様に宣言的かつ形式的な仕様記述により表現可能となる。しかし，サービス実現のためには，単にサービスの状態間の遷移を実現するサービスシナリオ（サービスロジック）を形式的に仕様を記述しても十分ではない。あるサービスを実現するためには他のサービスとの競合などの（静的・動的の両面での）関わりを形式的に記述可能な能力が必要である。

状態遷移に関わる各種オブジェクトの着脱は，本研究で提案する仕様化からプラグインまで自動化の範囲は非常に広がる。しかし，前述したサービス分析に関わる分析トリーの作成には人手に依存せざるを得ない。

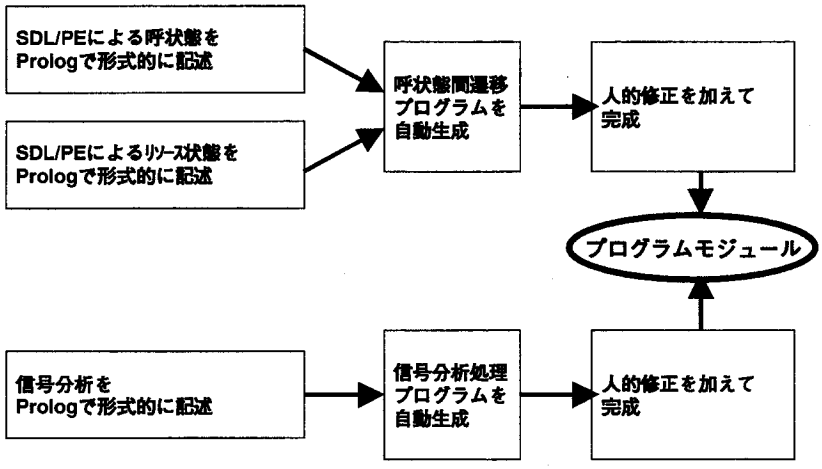


図 75 計算機支援設計

第7章 結論

【関連学術論文　【A2】【A3】【A4】【A5】【A6】】

第1章で本研究の狙いとそれを達成するための各課題を紹介し，第2章以降その課題を解決すべき技術を提案すると共に，各章毎にそれら研究結果についての分析及び評価を紹介した。本章では各章毎に提案した内容を総括する。

7.1 ｵﾌﾞｼﾞｪｸﾄ指向通信ｿﾌﾄｳｪｱ構成技術の提案

通信システムのモデル化、階層化技術及び設計法について述べると共にオブジェクト指向通信ソフトウェア構成技術を提案した。これにより複雑なシステムの代表格である大規模な通信システムを把握可能な単位まで抽象化レベルを落とすことが可能となる。更に，第2章で提案しているプラグイン・アウト技術への親和性も優れており，オブジェクト単位のサービス・機能追加の着脱が容易になることを示した。また，サービスプログラムの設計が物理的なシステム構成に依存しない形で実現できる環境を提供可能になることから，サービスプログラム設計に向けた形式的仕様記述の適用範囲が大幅に増大することを示した（第6章と関連)。

次に説明するプラグイン機構と連携することによるソフトウェア生産性の向上は大きく，所内実験機による評価結果では，サービス階層プログラム（アプリケーションプログラム）は2倍の生産性向上が，リソース階層及び実行制御階層を含めたトータルな生産性は15%の向上を得ることができた。これらの評価は，NTTで機能追加・変更した過去2年間のサービスを全て本システムに追加・変更して評価したものである。

更に，この様な効果を処理増で払った税金は処理時間で数%増，システム初期設定時間で数%増の範囲で実現可能である。

この結果を踏まえ，本研究が提案する通信ソフトウェア構成技術がNTTのこれからの交換システムの全てに適用されることが決定され，1996年12月に導入されたPHSサービス向け大容量交換システムを皮切りに，1997年導入予定の大容量ISDN交換システム及びこれから導入されるATM交換システムに適用される。

7.2 プラグイン・アウト構築技術の提案

第 2 章で紹介したオブジェクトを運用中の通信システムに着脱可能とするプラグイン・アウト技術を提案した。これによりサービスを受けているユーザに影響なく運用中の通信システムにサービス・機能の追加･変更が可能になる。リアルタイム性と高信頼性を踏まえた上でオブジェクトの追加･変更ができることにより，従来のシステムファイルによるシステム立ち上げをまたずにサービス・機能の追加ができることから，今後の通信システム構築における必須機能と考える。

前項で述べた様に本技術は，所内試験で評価され，その後，1996 年 12 月に導入された大容量 PHS で実際に運用で評価された。その結果，導入後 2 週間でバグフィックスを含めた機能追加・変更 200 件（メモリ規模で数メガ）に何ら問題なく活用でき，提案したプラグイン機構の信頼性と有効性，ファイル管理性の容易性さらにサービス・機能追加の即応性が確認された。従来の方式では，この様な短時間でこれだけの規模の追加・変更は困難であり，過去に実施した経験がない。大きな変革として評価できる。

7.3 共通ﾌﾟﾗｯﾄﾌｫｰﾑ構築技術の提案

第 3 章で提案したプラグイン・アウト機能を多種多様な通信システムに適用するための共通プラットフォーム化技術を提案した。更に，システムの分散化をサービスプログラム設計者に意識させない言語レベルの分散 OS 技術についても提案した。

言語モデルが提供するタスキング機能をシステム分散制御にまで発展させることにより，サービスプログラム設計者は，タスク間通信を提供する各種システムコールを意識することなくサービスプログラム設計を行うことができる。

前者のプラグイン・アウト機能の共通プラットフォーム化については，まず所内試験で各種サービスを提供する通信システムに適用し，その実現性が得られたことから，1996 年 12 月に導入された PHS サービス向け大容量交換システム，1997 年導入予定の大容量 ISDN 交換システム及び ATM 交換システムなどの各種通信システムへの適用が実施されている。

後者で述べる言語レベルの分散 OS 技術の適用については，所内実験で評価された 20%の処理増は，商用システムへの適用を考えると経済的でないことから所内実験限りとなった。しかし，本技術の適用により全く異なる交換システムとのサービスプログラムの流通実験を行い良好な結果を得ることができた。言語レベルでシステム

依存機能を完全に隠蔽できたため，NTT の所内実験システムと全く異なるアーキテクチャを持つ海外ベンダの交換システム上で動作するサービスプログラムの流通性を確認できた。ただし，システム制御，特に呼救済のコンセプトが異なっているためにシステム制御についてはサポートしていない。つまり，IT 分野で提供されているレベルのシステム再開のみをサポートし，交換特有の高信頼性を追求した呼救済に関わるシステム制御の実装は対象外とした。交換レベルのサービスプログラムの流通を実現するためには，交換システムに異状が生じた場合に提供中のサービスをどのように扱うかの統一したコンセプトの確立が必要である。

7.4 システム及びネットワーク管理技術の提案

第 2 章から第 4 章提案される機能を実現する多様な通信サブシステムとその通信システムから構成されるビルディングブロック型分散システムへの TMN 適用に向けたプロトコルプロファイルを明らかにした。これにより、本論文で提案するプラグインイン・アウト処理に必要となる各種コマンドや自律メッセージを効率よく実行可能となる。

7.5 開発支援環境技術の提案

本研究で提案する通信ソフトウェアの開発に向けた開発支援技術について提案した。オブジェクト単位のサービス・機能追加の着脱とプラグイン・アウト機構を実現するプラットフォームの共通化の両技術を支援する技術を提案している。第 2 章及び第 3 章で示した生産性の向上は，ここで示した，支援技術に負うところが大きい。

また，物理的なシステム構成にフリーなサービスプログラムの設計支援技術についても提案した。本研究で提案している階層化構成によりサービスプログラムは論理的なリソースオブジェクトを如何に処理するかに着目すればよい。その結果，本研究で提案している様に宣言的かつ形式的な仕様記述により表現可能になる。しかし，サービス実現のためには，単にサービスのある状態間の遷移を実現するサービスシナリオ（サービスロジック）を形式的に仕様を記述しても十分ではない。あるサービスを実現するためには他のサービスとの競合などの（静的・動的の両面での）関わりを形式的に記述可能な能力が必要である。

状態遷移に関わる各種オブジェクトの着脱は，本研究で提案する仕様化からプラグインまで自動化の可能性は非常に高いが，前述したサービス分析に関わると人手に依存せざるを得ない。

参考文献

【1】John December, “Presenting JAVA: An Introduction to JAVA and HOTJAVA”, Sams.net 1995

【2】Davic Chappell, “Active X and OLE: A Guide for Developers & Managers”, Microsoft Press February, 1997

【3】Microsoft, “Microsoft Systems Management Server”, Net & Com February, 1997.

【4】池谷寛、新井満男、村田忠明、"電子交換機", オーム社、1971

【5】城水元次郎、"DEX-1 号機の呼処理プログラム"、日本電信電話公社電気通信研究所研究実用化報告第 16 号第 11 号、1967, pp. 2307-2340.

【6】Dahl, O., Dijkstra, E., and Hoare, C.A.R., "Structured Programming", London, England, Academic Press, 1972.

【7】Military Standard Ada Programming Language, ANSI/MIL-STD-1815A, American National Institute, Inc., 1983.

【8】CCITT, "Recommendation Z.200 CCITT High Level Language (CHILL)"

【9】松尾勇二、丸山勝巳、"交換プログラミング言語 CHILL", 電気通信協会, 1985, 1986（改訂版）。

【10】井上修、他、"D10 型自動交換機用 D100B プログラムシステムの概要", 施設（NTT）、Vol.32, No.11~No.12, 1980.

【11】Keister, W. et. al., "NO.1ESS: System Organization and Objectives", Bell System Technical Journal, September 1964.

【12】Hayward, W.S., "The 5ESS Switching System", Bell System Technical Journal, Vol.64, No.6, 1985.

【13】Spencer, A.E. et. al., "Special Issue on No.4 ESS", Bell System Technical Journal, Vol.56, No.7, September 1977.

【14】武田英夫、保坂努、吉岡正紀、"ディジタル中継交換機の制御方式", 日本電信電話公社電気通信研究所研究実用化報告書、Vol.28, No.7, 1979.

【15】渡辺晴光、河辺候一、斉藤潔、"ディジタル加入者線交換機のプログラム構成", 日本電信電話公社電気通信研究所研究実用化報告書、Vol.31, No.11, 1982.

【16】"Building Intelligent Network", IEEE Communications Magazine, Vol.26, No.12,1988.

【17】Shigehiko Suzuki, "IN Rollout in Japan", IEEE Communications Magazine, Vol.31, No.3, March 1993.

【18】NTT Review, "[特集] 高度インテリジェントネットワーク", Vol.5, No.5, September 1993.

【19】秋山稔、"近代　通信交換工学", 電気書院、1979.

【20】Bjarne Stroustrup, "The C++ Programming Language (second edition)" Addison-Wesley（和）　プログラミング言語 C++（第 2 版）トッパン

【21】 Grady Booch, "Object Oriented Design with Applications", The Benjamin/Cummings Publishing Company, Inc., 1991.

【22】 Books, F, "No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer, vol. 20(4), April 1987.

【23】 CCITT Recommendation X.710 (1991) | ISO/IEC 9595, Information Technology - Open Systems Interconnection - Common Management Information Service Definition.

【24】 CCITT Recommendation X.711 (1991) | ISO/IEC 9595, Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification.

【25】 大鐘久生、"SNMP と CMIP　TCP/IP と OSI ネットワーク管理",株式会社ソフトリサーチセンタ

【26】 Preliminary Specification: The Common Object Request Broker: Architecture and Specification, X/Open in conjunction with OMG

【27】 James Rumbaugh, "Object-Oriented Modeling and Design", Pretice Hall, Inc.,1991. (和)　オブジェクト指向方法論 OMT　モデル化と設計　トッパン

【28】 Guttag, J., "Abstract Data Types and the Development of Data Structure, in Programming Language Design", New York, NY, Computer Society Press, 1980.

【29】 Goldberg, Robson, "Smalltalk-80: The Language and Its Implementation", Addison-Wesly, 1983.

【30】 Parnas, D., "Why Software is Unreliable", Software Aspects of Strategic Defense Systems, Victoria, Canada: University of Victoria, Report DCS-47-IR.

【31】 CTRON 大系、"カーネルインタフェース"、オーム社

【32】 CCITT Recommendation Z.323, "Man-Machine Interface".

【33】 CCITT Recommendation Z.100-Z.104.

【34】 W. F. Clockin and C. S. Mellish, "Programming in Prolog", Springer-Verlag Berlin Heidelberg 1981.

発表論文

A: 学術論文誌

【1】Koyanagi, K., Hane, K., and Suzuki, T., "Boundary Conditions between Current Mode and Thermal Mode Second Breakdown in Epitaxial Planar Transistors", IEEE Trans. Electron Devices, Vol. ED-24, no. 6, pp. 672-678, June 1977.

【2】Koyanagi, K., Sunaga, H., Yamada, T., Tomura, M., and Kurihara, N., "Distributed Communications System Technology", IEICE Trans. Communications., vol. E77-B, no. 11,pp. 1350-1362, Nov. 1994.

【3】Koyanagi, K., Yamada, T., Sunaga, H., Okamoto, A., and Monden, H., "Non-Stop Service-Enhanceable Communications Software Platform Based on an Object-Oriented Paradigm", IEICE Trans. Communications, vol.E78-B, no. 7, pp. 1043-1055, July 1995.

【5】Ueda, K., Inamori, H., Sunaga, H. and Koyanagi, K, "Applying TMN to a Highly Reliable Distributed Switching Node", IEICE Trans. Communications, vol. E78-B, no. 1, pp. 24-30, Jan. 1995.

【6】Koyanagi, K., Saitoh, T., Kanada, T., and Ikeda, H., "Internetworking Technologies for the Multimedia Network", IEICE Trans. Communications, vol.E80-B, no.10 , pp.1386-1392 , Oct. 1997.

【7】Koyanagi, K., Sunaga, H., Yamada, T., and Ikeda, H., "Applicability Evaluation of Service Feature Enhancement using Plug-in Modification Technique", IEICE Trans. Communications (出版予定)

B: 著書

【1】著者：小柳　恵一、監修：石川 宏"C++：オブジェクト指向設計", 電気通信協会,オーム社 1994 年 7 月 1 日発行.

C: NTT 社内機関誌及びシンポジューム

【1】小柳恵一，他，"ディジタル加入者線交換機の制御方式"，NTT 研究実用化報告 vol. 32 no. 11 (昭和 58 年)

【2】Koyanagi, K., etc, "Control System and Software for Digital Local Switching System", Review of the ECL vol. 32 no. 2 (1984)

【3】丸山勝巳，小柳恵一，中村秀文 "交換機シュミレーション構成技術"，NTT R&D vol. 39 no. 7 1990

【4】小柳恵一，"ｺﾝﾋﾟｭｰﾀﾈｯﾄﾜｰｸ:高度化技術と将来構想"，NTT International Symposium'96, October 22, 1996.

【5】小柳恵一, 斎藤孝文, 渡部直也, 金田哲也, “高度化技術と将来構想”, NTT R&D vol. 46, no. 1 1997

【6】Koyanagi, K., Saitoh, T., Watanabe, N., and Kanada, T., "Technology Initiatives for the Near Future", NTT Review 1997.

D: 国際学術会議

【1】Koyanagi, K, Maruyama, K., "Declarative Specifications in Prolog and SDL", SDL'87; State of the Art and Future Trends, Proc. of the Third SDL Forum, The Hague, The Netherlands, April 1987.

【2】Kai, T., Maruyama, K., and Koyanagi, K., "The Program Structure Based on Current Object Model for Switching Systems", in Proc. SETSS'89, pp.34-38, June 1989.

【3】Maruyama, K., Watanabe, N., Koyanagi, K., and Kai, T., "A Concurrent Object Oriented Switching Program in Chill", in Proc. IEEE ISS'90, May 1990.

【4】Koyanagi, K., Shimizu, T., and Tomita, S., "Hierarchically Structured Switching Software", in Proc. IEEE Globecom'93, no. 3, pp. 1918-1933, Nov. 1993.

【5】Okamoto, A., Sunaga, H. and Koyanagi, K., "Dynamic Program Modification in the Non-Stop Extensible System (NOSES)", in Proc. IEEE SUPERCOM/ICC'94, pp. 1779-1783, May 1994.

【6】Sunaga, H., Okamoto, A., Yamada, T., and Koyanagi, K., “A Reliable Communication Switching Platform for Quick Service Provisioning”, in Proc. IEEE Globecom'94, pp.77-82, Nov. 1994.

【7】Yamada, T., Sunaga, H., Koyanagi, K., and Matsumura, H., "NOSES Object-Oriented Switching Software in C++", in Proc. IEEE Globecom'94, pp. 828-832, Nov. 1994.

【8】Inamori H., Ueda K., and Koyanagi K., "NOSES as a Key Technology for Realizing Advanced Telecommunications Management", Proceedings of IEEE Globecom'94, pp. 212-216, Nov. 1994.

【9】Matsumura, H., Sunaga, H., Koyanagi, K., and Yamada, T., "Highly Reliable On-line Partial File Modification For Office Data Provisioning in NOSES", in Proc. IEEE Globecom'94, pp. 1238-1242, Nov. 1994.

【10】Yamada, T, Koyanagi, K., Sunaga, H., and Matsumura, H., "Reliable Real-Time Software Structure for Distributed Switching Nodes", in Proc. IEEE Globecom'94, pp. 495-499, Nov. 1994.

【11】Inamori H., Ueda K., Koyanagi K., Sunaga, H, and Kishi, T., "Applying TMN to a Distributed Communications Node System with Common Platform Software", in Proc. IEEE ICC'95, pp. 83-87, June 1995.

【12】Sunaga, H., Okamoto, A., Yamada, T., and Koyanagi, K., "A Reliable Communication Switching Platform for Quick Service Provisioning", in Proc. IEEE ICC'95, pp. 77-82, June 1995.

【13】Sunaga, H., Ueda, K., Koyanagi, K., Okamoto, A., and Inamori, H., "File Operation for a Non-Stop Service Enhanceable Software (NOSES) Platform", in Proc. IEEE Globecom'95, Nov. 1995.

【14】Okamoto, A., Sunaga, H., Koyanagi, K., et al., "NOSES Plug-in Platform for Quick Service Provisioning", IEICE APCC '95.

【15】Sunaga, H., Tomita, S., Koyanagi, K., and Inamori, H., "Customer Network Management Service and MO Architecture for Data Networks", IEICE APCC'95, pp. 790-794, June 1995.

【16】Sunaga, H., Yamada, T., Koyanagi, K., and Sunaga, S., "Applicability Evaluation of Servoce Feature Enhancement using the Partial-File "Plug-in" Modification Technique", in Proc. IEEE ICC'96, pp. 32-36, June 1996.

【17】Koyanagi, K., "Open Computer Network: OCN/OCNe", 6th ATM/B-ISDN Technical Workshop, September 1996.

【18】Koyanagi, K., Terayama, T., Adachi, T., and H. Sugioka, "Internetworking Technologies for GMN-1 (Global Mega-media Network-1)", APCC'97, Sydney, December. 1997.

【19】Yamaguchi, H., Adachi, T., Koyanagi, K., and Hagishima, K., " Strategy for GMN (Global Mega-media Network)", APCC'97, Sydney, December.

E: IEEE Communication Magazine

【1】Maruyama, K., Watanabe, N., Koyanagi, K., Kai, T., and Tomita, S., "A Concurrent Object Oriented Switching Program in CHILL", IEEE Communications Magazine, vol. 29, no. 1, pp. 60-68, Jan. 1991

【2】Koyanagi, K., Saitoh, T., and Kanada, T., "Technology Initiatives for the Near Future in Internetworking", IEEE Communications Magazine, May 1997.

F: その他の機関誌及び解説記事

【1】小柳恵一, "通信ソフトウェアのオブジェクト指向設計", 電子情報通信学会誌, vol. 79 no.1, 1月(1996)

G: その他の講演

【1】小柳恵一, "OCNの高度化技術と将来構想", NET&COM'97, 2月(1997)

【2】Koyanagi, K., "Network Services in Multimedia Era", AEU(Asia Electronics Union) in Korea, May 1997

H: 全国大会

【1】小柳恵一，他，“ディジタル加入者線交換機における機能配分の考察”，電子通信学会 (昭和 53 年)

【2】小柳恵一，他，“ディジタル加入者線交換機の障害処理方式”，電子通信学 301 (昭和 54 年)

【3】小柳恵一，他，“NO. 7 信号方式のユーザパート構成法の検討”，電子通信学会 1608 (昭和 57 年)

【4】小柳恵一，他，“ディジタル加入者線信号に関するプログラム処理”，電子通信学会 250 (昭和 57 年)

【5】小柳恵一，他，“INS 加入者線交換機における IOT 分離方式の検討”，電子通信学会 1786 (昭和 57 年)

【6】小柳恵一，他，“INS 用呼制御方式に関する一考察”，電子通信学会 (昭和 58 年)

【7】小柳恵一，丸山勝巳，“SDL と Prolog による宣言的仕様記述の一考察”，電子情報通信学会，N0. 1891，昭和 62 年

【8】渡部信幸，片山悦子，小柳恵一，“分析処理への RDB の適用”，電子情報通信学会，NO. 5341，昭和 62 年

【9】須永宏，小柳恵一，渡辺一男，“パケット交換へのオブジェクト指向モデル適用法の一考察”，電子情報学会，NO. 2632，昭和 62 年

【10】片山悦子，甲斐俊洋，小柳恵一，“オブジェクト指向型交換サービス制御方式の検討”，電子情報学会，B－319 (1989)

【11】山田哲靖，渡部信幸，小柳恵一，“オブジェクト指向型交換リソース制御インタフェースの検討”，電子情報学会，B－320 (1989)

【12】中村秀文，甲斐俊洋，渡部信幸，小柳恵一，“オブジェクト指向概念を用いた通信プロトコルインプリメント法の検討”，電子情報学会，B－547 (1991)

【13】松村一，須永宏，春日俊六，小柳恵一，“機能追加を考慮した所データ変更方法の検討”，電子情報学会，B－540 (1991)

【14】岸達也，稲守久由，山田哲靖，上田清，小柳恵一，“TMN を適用した通信ノード用ソフト流通支援環境の一検討”，電子情報学会 (1994)

【15】稲守久由，上田清，岸達也，小柳恵一，“TMN を適用した分散型通信ノードシステムの一検討”，電子情報学会 (1994)

【16】石橋竜也，須永宏，小柳恵一，“TMN による局データ管理方式の検討”，電子情報学会 (1994)

I: 研究会

【1】小柳恵一，他，“ディジタル加入者線交換機における障害処理方式”，電子通信学　SE80-51 (昭和 55 年)

【2】小柳恵一，他，“サービス総合交換機における中継線インタフェースの検討”，電子通信学会　SE82-12 (昭和 57 年)

【3】甲斐俊洋，丸山勝巳，小柳恵一，“実時間システム用並列オブジェクト実行モデルの考察”，電子通信学会，交換研究会，SE86－111，1986

【4】小柳恵一、丸山勝巳、甲斐俊洋、"論理構造を反映したオブジェクト指向型交換プログラム構成法の考察"，電子通信学会　交換研究会、SE86-133, 1987

【5】甲斐俊洋，丸山勝巳，小柳恵一，渡部信幸“オブジェクトモデルによる交換プログラム構成”，電子情報通信学会，交換研究会，SE87－147，1988

【6】渡部信幸，小柳恵一，丸山勝巳，“実時間オブジェクト指向交換プログラム構成の考察”，電子情報通信学会，交換研究会，SSE88－83，1988

【7】片山悦子、甲斐俊洋、小柳恵一、"交換サービス制御へのオブジェクト指向プログラム適用法の考察"，電子情報通信学会 SSE89-77

【8】小柳恵一，“分散構成ノードへのオブジェクト指向型交換プログラムの適用”，電子情報通信学会　SSE89－164

【9】岡本明、須永宏、小柳恵一、"運用中ファイルの更新についての一考察"，電子情報通信学会 SSE92-26

【10】小柳恵一，須永宏，山田哲靖，“分散環境のインパクト:分散通信システムの主要技術”，電子情報通信学会，ネットワーキングアーキテクチャワークショップ　(1993)

【11】岸達也，稲守久由，山田哲靖，上田清，小柳恵一，“TMN 適用新通信ノードのソフト流通支援環境の一検討”，電子情報通信学会　SSE93－21

【12】岡本明、山田哲靖、須永宏、小柳恵一、“NOSE S 実現のためのプログラム記述規定に関する考察”、電子情報通信学会 SSE94-141