■5群（通信・放送） － 4編（ノード技術）

4章 電話／データ統合システム

■概要■

【本章の構成】

■5群 － 4編 － 4章

4-1 回線交換とデータ交換の統合型ノードシステム

（執筆者：片山悦子）［2009年4月 受領］

交換ノードシステムは，多くの要求条件に応えるために，非常に大規模になっている．多種多様なサービスの実現とその機能追加性，ユーザに遅延を感じさせない実時間性，多くのユーザを収容し同時にサービスを提供する多重性，障害時にもすぐに復旧し影響を最小化する信頼性，障害を発生させない高品質，頻繁な修正や機能追加に即応できるファイル／ハードパッケージ入れ替え機能，24時間ノンストップ運転，収容情報・接続情報の変更の容易性など，数千万加入者が通信し合うネットワークを実現するシステムには，キャリアグレードとして様々な要求条件が課せられている．

1990年代前半までのノードシステムでは，回線交換，データ交換，それぞれに固有のハードウェア，固有のソフトウェアがあり，異なる構成・実現方法で開発されてきた．しかし，要求条件が厳しくなり，開発量が増大していくなか，効率的な開発が求められるようになった．要求条件の基本は共通であり，統一したアーキテクチャとすることにより，効率的な開発ができると考え，ISDN，アナログ，パケット，広帯域系などの各ノードシステムに共通に適用可能なアーキテクチャをもつ, MHNと呼ばれるノードシステムがNTTにて開発された．1996年に商用化され，2009年現在に至るまで運用に供している．

MHNシステムシリーズには，加入者収容回線交換ノードのMHN-S，パケット交換ノードのMHN-P，フレームリレーノードのMHN-F，ATM交換ノードのMHN-A，高機能INサービスを実現するためのMHN-SCPなどがある．以下に，先行開発されたMHN-Sを中心にアーキテクチャを紹介する．

4-1-1 アーキテクチャ

各ノード種別ごとに特有のハードウェア，プロトコルなどを隠蔽するために階層化構造のアーキテクチャになっている．各階層の位置づけ・機能を明確にし，各階層間のインタフェースを明確に定義している．それにより，ハードウェアのマルチベンダ化によりプロセッサ置換を実現したり，各ノード種別共通にシステム管理機能を提供したりすることができる．階層は，ソフトウェア5階層とハードウェア階層からなる．ソフトウェア5階層は以下のとおりである．

- **呼処理サービス階層**：サービス呼状態を管理し，個々のリソースを制御してサービスを実行する．サービスごとの処理を実行するサービスシナリオ，サービスを決定するための各種分析機能などから構成される．
- **呼処理リソース階層**：通話路構成，信号装置構成などのシステムアーキテクチャ，プロトコルを隠蔽する．各種プロトコル制御，通話路制御，加入者線／中継線制御などの機能から構成される．
- **保守運用階層**：システムを保守運用するための機能を提供する．サービスオーダー，トラヒック制御，トラヒック測定，試験などの各機能から構成される．
- **共通プラットフォーム階層**：各ノードに共通なシステム機能を提供し，システムの冗長構成，システム異常時の動作を隠蔽する．システム管理，装置管理などから構成される．

ノードシステムのキャリアグレードを実現するための，障害処理，再開機能，ファイル更新，プロセッサ置換などの機能を提供する．

- **実行制御階層**：ハードウェアアーキテクチャを隠蔽する．カーネル，各種ドライバから構成される．

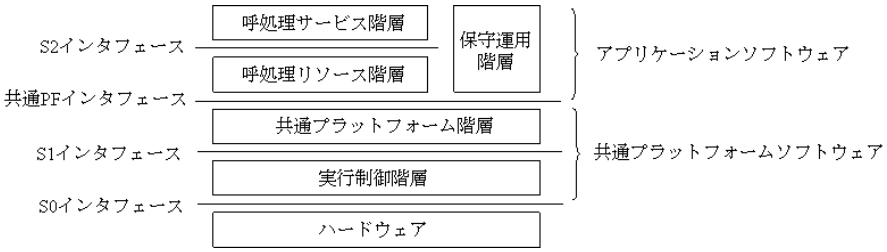


図 4・1　アーキテクチャ

階層間インタフェースは，階層ごとの差替えが可能なように，下位階層の機能を隠蔽している．実行制御階層・共通プラットフォーム間の S1 インタフェースは IROS 準拠のインタフェースになっている．リソース階層-サービス階層間の S2 インタフェースは，UNI/NNI などを参考にしているものの，論理性の高い疎結合なインタフェースである．
共通プラットフォームソフトウェアは，MHN シリーズで共通であり，アプリケーションソフトウェアは，ノード種別ごとに開発されている．

4-1-2 アプリケーションソフトウェア構成技術

(1)　オブジェクト指向設計

プログラムの理解性を向上させるため，大規模プログラムの開発に適しているといわれていたオブジェクト指向設計を採用している．ノードシステムをモデル化し，モデル化された各論理要素（加入者線，中継線，呼，加入者契約情報，サービス回路，音源など）をオブジェクトとしてとらえたモジュール化手法をとっている．

各オブジェクトは，保有データとそれに対する手続きがカプセル化されたものであり，その定義はクラスと呼ばれる．オブジェクトの実態をインスタンスと呼ぶ．実際のオブジェクトの状態，データ設定値により，一つのクラスから複数のインスタンスが生成される．例えば，中継線をクラスとして定義し，個々の回線対応にインスタンスが生成されることになる．インスタンスは，常時システムに存在するものはシステム初期設定時に，要求ごとに存在するものは，要求が発生したときにのみ生成（捕捉）される．

各オブジェクト間はメッセージで通信し合う．データに着目した論理要素単位にモジュール化されていることから，メッセージは疎結合なインタフェースとなっている．

各クラス間で共通な機能や類似の機能がある場合，クラス間に上下関係をつけ，上位クラス（基底クラス）の機能を下位クラスが継承することができる．これをインヘリタンスと呼ぶ．インヘリタンスは有効な部品化手法である．

C++言語が普及した現在，オブジェクト指向プログラムは目新しいものではないが，当時，

大規模で実時間性の高い商用システムにいち早く採用し有効性を示したという点で，意義あるシステムである．このようなモジュール化手法により，機能が異なる各ノードシステムで同様の設計思想が適用でき，更には，クラス単位での着脱挿入（プラグイン）などの有用機能の実現を容易にしたといえる．

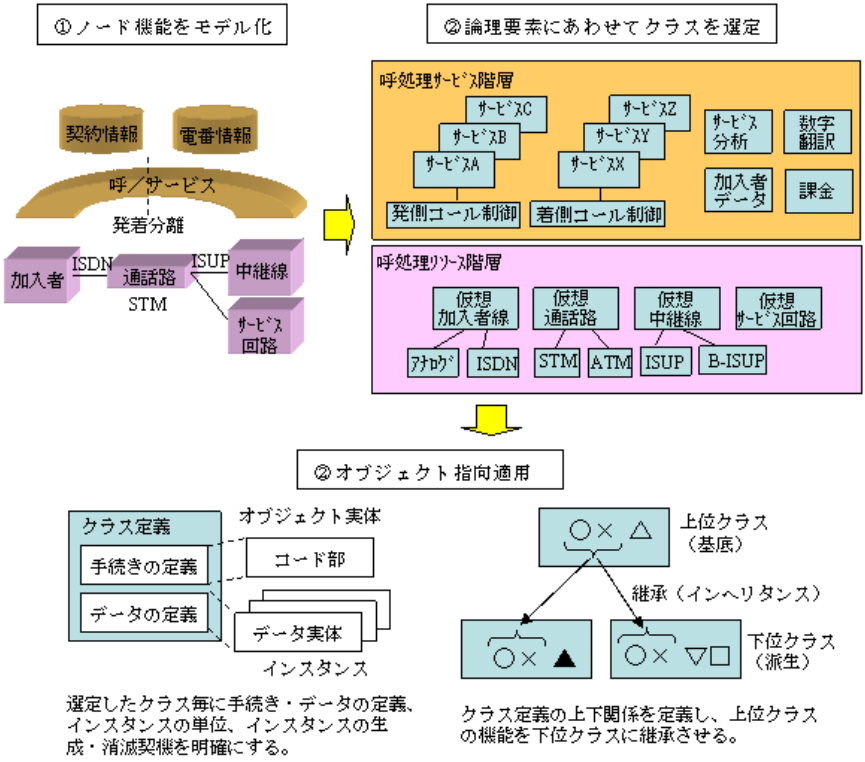


図 4・2　オブジェクト指向適用法

(2)　サービスシナリオ方式

ノードシステムが提供するサービスは多種多様であり，それらサービスが互いに影響しあう場合が多い．特に，加入者を収容するノードでは，加入者の呼状態，加入者の契約条件，サービス間の競合条件，受信イベントなど，様々な情報を分析することにより，提供するサービスが決定されるため，サービス呼状態遷移とサービス分析が複雑になっている．この複雑性を解消し，サービスの機能追加を，既存サービスに擾乱を与えず，極力他のサービスから独立して開発するために，サービスを体系的に整理し，実現方法をパターン化したのが，サービスシナリオ方式である．サービスシナリオ方式では，「サービス」を論理要素ととらえ，サービスの状態遷移や固有処理をサービスシナリオと呼ぶオブジェクトとして定義している．

サービスは，通話路制御などを伴い状態が遷移する接続系サービスと，データ検索やデー

タ変換だけの状態が遷移しない論理系サービスに分類できる．また，接続系サービスは，発側と着側の二者間の通信だけの二者通信サービスと，コールウェイティングや三者通話のように多くの通話者が割り込み合う多者通信サービスがある．このようにサービスを分類し，それぞれの実行形態をパターン化している．

サービスシナリオは，サービス種別，起動契機，契約条件などにより選定されるため，非常に多くのサービスシナリオが定義されるが，類似のサービス間の共通機能は，サービスシナリオ間でインヘリタンス関係をつけることにより，効率よく開発することができる．

また，サービスを決定するためのサービス分析は，電話番号，加入者データなどのデータに着目したオブジェクトが行うことにより，サービス分析の単位が小さくなり，機能追加箇所も局在化され，理解性の高いサービス分析を実現している．

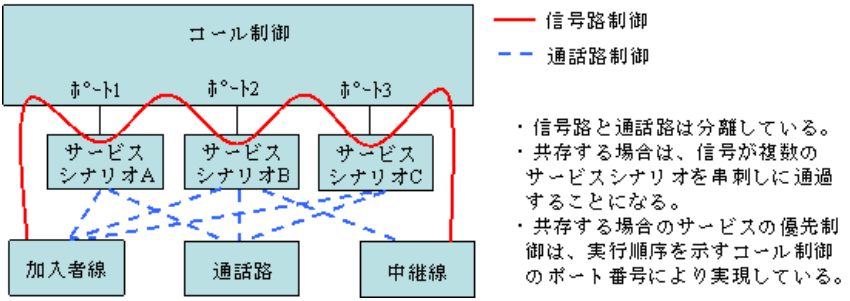


図 4・3　サービスシナリオ方式

(3)　タスク適用方式

ノードシステムでは，多数の通信を同時に扱うため，信号の競合，データ書き込み中の割り込みなど，並列処理や排他制御を確実に行う必要があり，その設計は重要である．並列処理を実現するために OS がもつタスク機能を使用しているが，タスク割当て単位，タスク生存期間などは，実行性能やメモリ量も考慮したうえで，適切に設計を行わなければならない．このシステムでは，1 イベント 1 タスク方式を基本とし，呼，加入者線，中継線などのオブジェクトにタスクを割り当てるとともに，オブジェクト間通信として，タスク乗せ換えの有無，排他制御の内外の組合せにより，タスク間通信／プロシージャコールの使用方法を明確にしている．

（出典：NTT R&D 誌, vol.45, no.6, pp.41-48）

■5群 － 4編 － 4章

4-2 STP/高度 IN サーバ

4-2-1 STP

（執筆者：村上龍郎）［2009年4月 受領］

交換ノードシステム間の制御信号（共通線信号）を転送する「共通線信号網」の交換機を共通線中継交換機（STP : Signaling Transfer Point）と呼ぶ．共通線信号網は独立した通信網であり，音声やデータコネクションと分離した通信経路上を転送され，共通線信号の転送により，交換ノード間のトランザクションサービスや呼制御が行われる．

共通線信号網は一つの信号リンクで複数交換機の通話回線を制御することから，高信頼度が要求される．このため網構成は，複数ルートを用意し，負荷分散を行い，かつ片系異常時でも残る系で全信号トラヒックを処理可能となり，低中継遅延時間も満足する設備量で構成される．

網内接続構成は，高信頼性を確保するために二重化構成とし，すべての発信信号局と着信号局間で2ルートの信号方路を確保する．信号区域相互間ではクロスルートを加えた4ルートの信号方路も考えられる．各々の信号方路は負荷を均等分散で運用し，故障時には残る正常な方路で全信号が処理可能となるよう設計する．二つの隣接信号局間の通信経路はリンクと呼ばれる．同じ組の隣接ノードを接続するリンクの集合はリンク群と呼ばれる．共通線信号網におけるリンク群種別は4種類である（**表 4･1**）．網内接続冗長構成例を**図 4･4** に示す．

表 4・1　共通線信号網における信号リンク群種別

種別		
基幹	ストレートリンク群	自信号区域外自面STP向けリンク群
	クロスリンク群	自信号区域外他面STP向けリンク群
面間リンク群		自信号区域他面STP向けリンク群
SEP向けリンク群		

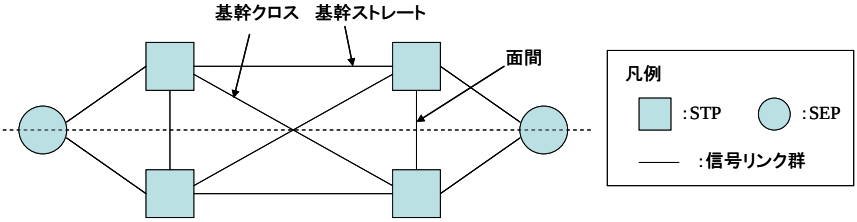


図 4・4　網内接続冗長構成例

(1) 信頼性を確保するため2面構成とし，各面をA面，B面とする．各SEP（Signal End Point : 信号端局）は，A･B両面のSTPに二重帰属とし，二つのSTPは同一信号区域（一対のSTPで構成されるエリア）を受け持つ．

(2) 故障時の他面への迂回を容易にするために，A，B面間で対となるSTP間に面間リンク

群を設ける.

(3) 二重故障などによる信号区域間途絶を防止するため,隣接する STP に対応する他面の STP に基幹クロスリング群を設ける.

●信号局番号

信号局のポイントコード（PC）は，各種通信サービスを提供するうえで，信号転送する際に必要となる信号端局識別，及び信号中継局識別のための番号で，普遍性・融通性・保守性を重視して計画する必要があることから，将来の需要変動及び新サービスの導入に対しても十分対応できる余裕をもった構成が必要である.

(1) ポイントコード区域　　網構成に合った PC 体系とするため，主番号区域（MNA：Main Numbering Plan Area of Signal），副番号区域（SNA：Sub Numbering Plan Area of Signal）を設定し，2 階層の PC 体系とする（**図 4･5**）. PC の付与は，複数の事業会社を含めた共通線信号網において一意の PC を付与する.

(2) SEP の需要に基づき，MNA（M），SNA（S），及び信号区域内ユニット（U）に番号を付与する.

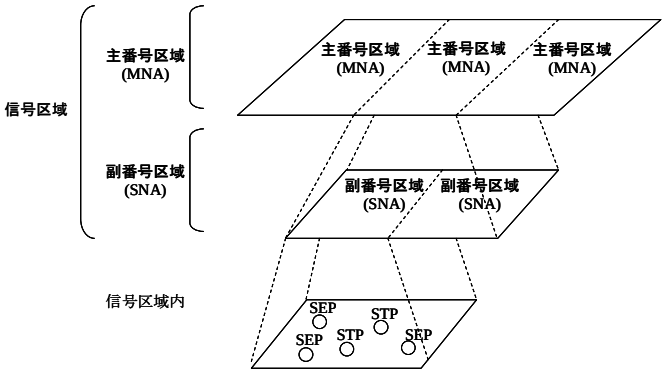


図 4・5　主番号区域と副番号区域の関係

ポイントコード構成を**表 4･2** に示す.

表 4・2　ポイントコード構成

B15 b14 b13 b12 b11 b10 b9	b8 b7 b6 b5	b4 b3 b2 b1 b0
区域内ユニット番号 (U)	**副番号区域 (S)**	**主番号区域 (M)**

4-2-2 高度 IN サーバ

（執筆者：中村俊郎・吉見正信）［2009 年 5 月 受領］

(1)　インテリジェントネットワーク（IN）

電話網は発信者のダイヤル情報に従って，着信者までの回線を順次接続するという基本的なサービスを中心に発展してきた．一方，通信サービスの高度化に伴いネットワークワイド

に番号変換を行ったりカスタマ個々の要望に合わせて接続形態を変更するような新しいサービス（フリーフォン（NTT 商品名：フリーダイヤル），情報料課金サービス（NTT 商品名：ダイヤル Q2），パーソナルハンディホンシステム（PHS）など）の構想が出現し，これらのサービスを効率的に実現するための新しいネットワーク構造が必要と認識されるようになった．この構造がインテリジェントネットワーク（IN）と呼ばれるもので，IN では，交換機自体は基本的な呼接続機能のみをもち，番号翻訳，位置情報管理など，高度でネットワークワイドなサービス制御機能は交換機とは別に設けたサービス制御ポイント（SCP）が分担する．これにより，高度で多様なネットワークサービスが実現可能となった．なお，IN という言葉は，米国の研究開発機能である Bellcore が最初に提唱したものであり，その後各国が独自の仕様でサービス開発を進めたことが高度 IN 構想につながっていくこととなる．

(2)　高度 IN（AIN）

IN は当初，交換機と SCP 間の機能分担やインタフェースについては標準化されておらず，各国独自の仕様でサービスを提供していた．日本では，データベースの検索機能のみを SCP に集中配備し，サービスに関する呼制御機能は交換機に配備する方式が採用された．このため，サービスを追加・変更するたびに交換機のソフトウェアにも手を加える必要があり，サービス開発を効率の悪いものにしていた．

そこで，交換機側は個々のサービスに依存しないように動作をモデル化し（**図 4･6**），データベースの検索機能のみならずサービスに関する呼制御機能を SCP に集中配備，また交換機・SCP 間のインタフェースをアプリケーションレイヤまで標準化した高度 IN と呼ばれるネットワーク構造が ITU-T において規定された（**図 4･7**）．

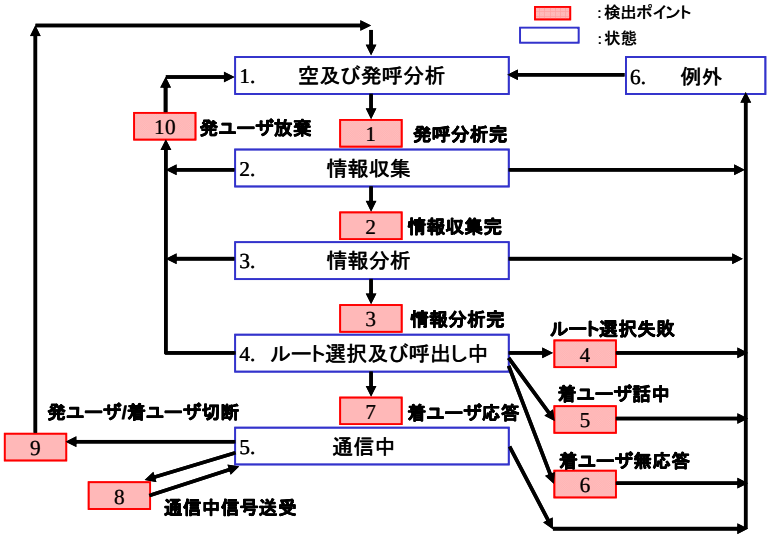


図 4・6　基本呼状態モデル（能力セット 1 の発側の例）

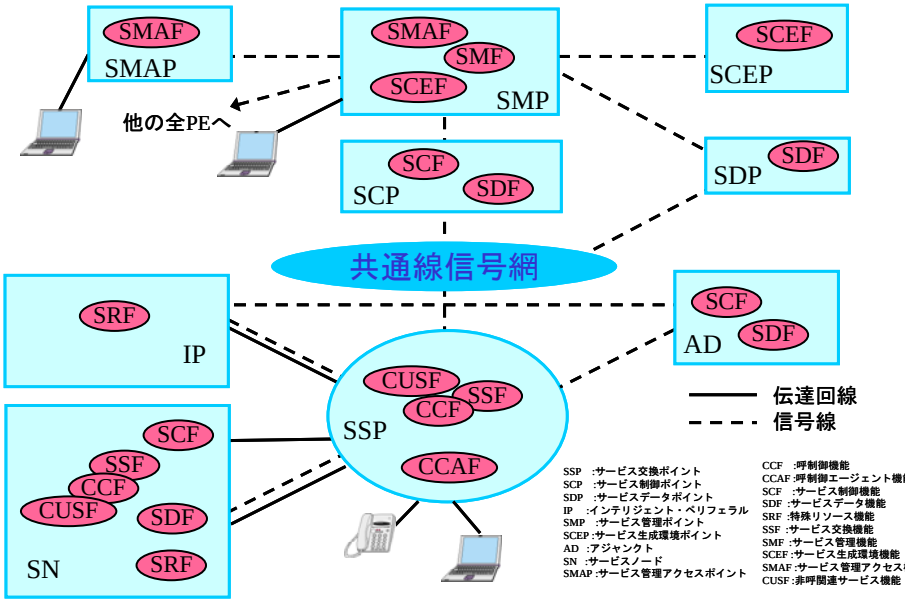


図 4・7　高度 IN のアーキテクチャモデル

高度 IN では新サービス対応の呼制御機能を SCP に配備することにより交換機（伝達ノード）と無関係に新サービスを提供できるようになった．ソフト面でも一般カスタマレベル，サービス開発レベル，システム開発レベルに階層化したほかに，画面による入力操作でソフトの自動生成を可能としサービスソフトの開発期間の短縮化を図った．更に，サービスの定義用端末からの簡単な入力によってサービスの定義付けや変更（カスタマコントロール）ができるようにしたほか，そのサービス仕様に関連するサービス制御ソフトを個々に作成し実現できるようにしたことにより，サービスのカスタマイズ化が完成した．高度 IN の登場でネットワークのオープン化が促され，タイムリーなサービスが提供できるようになった．

高度 IN を構成する主なノード(システム)には，以下に示すサービス制御ポイント(SCP)，サービス管理ポイント（SMS），サービス生成環境（SCE）があり，これらの連携により交換機であるサービス交換ポイント（SSP）を制御し高度 IN サービスを実現する．

(3)　サービス制御ポイント（SCP）

サービス制御ポイント（SCP）は，サービス制御機能（SCF）とサービスデータ機能（SDF）から構成され，高度 IN においてサービス制御を行う．具体的には，番号変換（フリーダイヤル番号から一般電話番号への変換など），交換機に対する各種接続制御指示・課金制御指示（発着どちらの課金にするか，課金レートをどうするかなど），運用情報の収集などを行う．高度 IN では，交換機-SCP 間，及び SCP-SCP 間インタフェースに，共通線信号方式の IN アプリケーションプロトコル（INAP）と呼ばれる国際標準インタフェースが適用されるので，

異なる通信事業者間でのサービスの連携，データの共有も容易になる．サービス制御では，サービスに共通な機能部品を用意し，これらの組合せでサービス仕様を実現する方法がとられる．このサービス仕様を実現するためのプログラムは，サービス論理プログラム（SLP）と呼ばれ，プログラミングやサービスの知識をもたない一般のカスタマ自らが記述できるカスタマレベル，詳細なサービス仕様の知識をもつサービス開発者レベル，プログラミングスキルとシステムの詳細な知識をもつシステム開発者レベルの三つに階層化することでサービス追加／開発時の影響範囲を局所化し，効率的な開発を実現している．

(4) サービス管理システム（SMS）

高度 IN において，サービス契約情報の管理及び SCP へのダウンロード，サービス仕様を記述した SLP の管理，及び SCP へのダウンロード，カスタマ制御機能の提供，運用情報の管理及びカスタマへの提供などを行うシステムがサービス管理システム（SMS：図 4･7 の SMP に相当）である．SMS はデータベースと通信を中心とした情報処理装置であり，一般的には汎用のコンピュータやサーバが使用される．

(5) サービス交換ポイント（SSP）

サービス交換ポイント（SSP）は，呼接続制御機能（CCF）及びサービス交換機能（SSF）を中心に構成される．CCF は図 4･6 に示した基本呼状態遷移モデル（BCSM：Basic Call State Model）に基づき呼の接続制御を実施し，AIN 呼を検出すると SSF に通知したり，SSF からの接続指示に基づき，通話路などの物理リソースを制御する．SSF は CCF からの通知に基づき SCP へ AIN 呼の起動などを通知する機能，及び AIN 呼の状態管理やサービス間の相互動作機能を有する．

(6) サービス生成環境（SCE）

高度 IN では，ネットワーク機能をサービス非依存な部品に分解し，これを組み合わせることにより，サービスを実現する概念整理が行われた．このため，SCP で走行する SLP は，自動生成可能な構造にすることができる．サービス生成環境とは，この特徴をいかしてソフトウェアの専門家でないサービス開発者が図形表現などのわかりやすいユーザインタフェースを利用してサービス仕様を記述し，容易に SLP を開発可能にするサポート環境をさす．

(7) カスタマコントロール

AIN では，サービス契約ユーザ（カスタマ）自身が，個々の希望に応じて，サービスの条件や論理を修正することが可能である．このようにカスタマがサービス内容を制御することを，カスタマコントロールと呼ぶ．制御可能なパラメータの例としては，接続先（転送先電話番号など），着信呼を複数箇所に分配する比率，着信許可する発信番号・地域・時間帯などがある．また高度 IN では，カスタマ自身が SLP の一部を簡便な図形言語で記述することにより高度なカスタマイズも可能である．

(8) 高度 IN により提供されたサービス

高度 IN で規定された標準化インタフェースを用い，様々なアプリケーションサービスが

開発されている．例えば，ユーザ個人単位に番号を払い出し，ユーザがどこにいてもその番号で通信を可能とする UPT サービス（NTT 商品名：e コールサービス）では，サービス制御に必要なネットワーク機能にはモビリティ制御，認証制御，網間接続，サービスプロファイル（契約条件）管理などがあり，これらは高度 IN により実現されている．このほか，PHS，フリーフォンサービス，ナビゲーションサービス，情報量課金サービス，IC カード公衆，発信者名表示などが高度 IN により実現されている．

■参考文献

1) 鈴木滋彦，石川　宏, “新ノードシステム（NS 8000 シリーズ）の実用化ーマルチメディア時代に向けてー,” 信学誌, vol.81, no.8, pp.789-815, Aug. 1998.
2) 電子情報通信学会編, “電子情報通信ハンドブック,” 7-8 編 交換システム, pp.904-916, 2000.

4-3 交換システムプラットフォーム

（執筆者：二神　新・山田哲靖）［2009 年 4 月 受領］

4-3-1 大規模ソフトウェア構造

統合ノードシステムのソフトウェアは，マルチベンダ環境を考慮して各システムの統一ソフトウェアアーキテクチャとして，標準 OS インタフェース IROS（Interface for Realtime Operating System）に基づく階層化ソフトウェア構造を採用した（**図 4･8**）[1]．S1 インタフェースは，ベンダ間のハードウェアの差を隠蔽する標準インタフェースである．これにより，上位のソフトウェアは，マルチベンダ環境においても，ベンダごとに複数ではなく，一つとすることが可能となった．

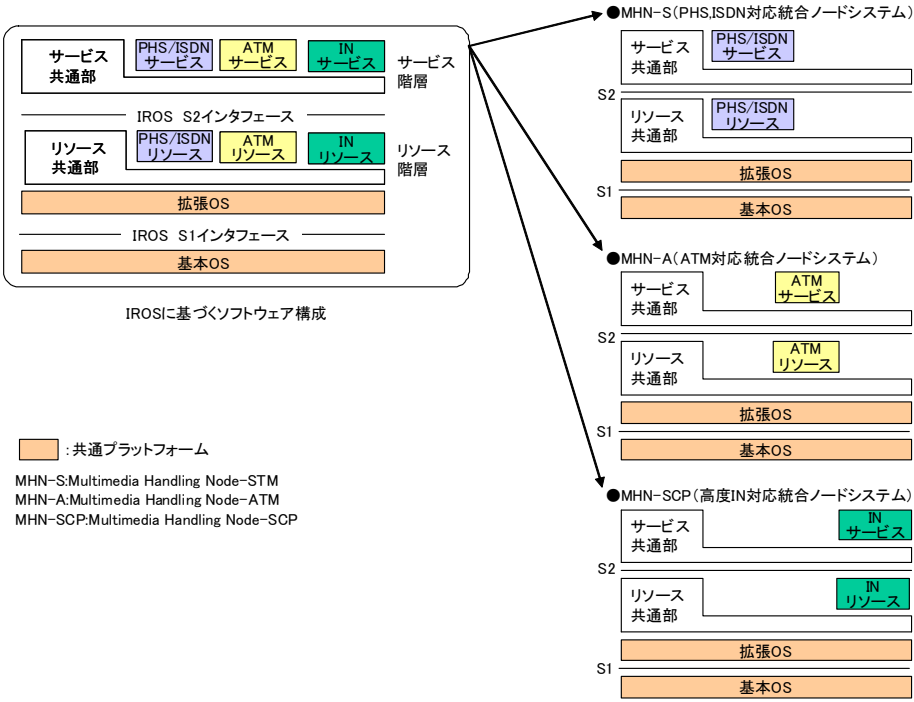


図 4･8　IROS に基づく共通プラットフォーム

S1 インタフェースと S2 インタフェースとの間の階層には，各種サービスを提供する際に共通的に用いられる共通プラットフォームならびにリソース階層を設けた．各システムソフトウェアは，S2 インタフェースより上のサービス階層に，PHS，ISDN，ATM，高度 IN などの各アプリケーションを配備して実現する．これにより，ソフトウェアの流通性を高め，ソフトウェア生産性を向上させた．

また，ソフトウェアの階層化に加えて，オブジェクト指向技術やプラグイン技術などを導

入することにより，新サービスの提供時や機能追加時の効率的なソフトウェア開発を可能とした．

4-3-2 リアルタイム OS

統合ノードシステムでは，実時間実行制御とハードウェアのドライバ機能などを提供するリアルタイム OS を開発した．本リアルタイム OS は IROS 仕様に準拠し，ベンダごとのハードウェアアーキテクチャ差の隠蔽と，ノードシステムに要求されるリアルタイム性と信頼性を実現している．従来システムと比較すると，論理空間制御機能により，実メモリ容量の制限がなくなっており，リング保護機能などによるメモリ保護（プログラム及びデータ保護）の強化が図られている．

論理空間は，共通域（システム一意の空間で，全タスクから共用可能）とユーザ域（共通域以外の空間で，複数存在可能）から構成される．ユーザ域をタスク独立の生成，削除などの制御対象とできるので，**図 4･9** に示すような 3 種類のタスクの対応法が考えられる．図 4･9（タイプ 1）の全タスクが一つのユーザ域上で走行するという従来システムと同様な方法では，メモリ保護がリングレベル及びアクセス権のみであるため，重要度の異なる種々のメモリ内容を保護するためには十分でない．図 4･9（タイプ 3）の情報処理でしばしば使用される多重仮想記憶方式では，メモリ保護を強化できるが，タスク生成／削除の処理オーバヘッドが大きいいため，呼処理などに適用することは困難である．よって，両者の長所を取り入れた（タイプ 2）の形態を採用した．

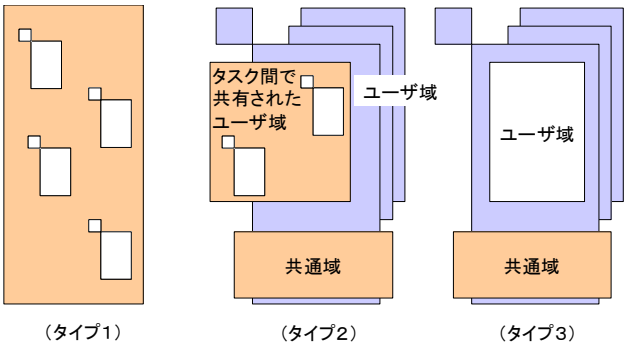


図 4・9　ユーザ域の共通形態

4-3-3 ソフトウェア設計技術

大規模ソフトウェアを短期間でかつ少ない稼働で開発するためにはソフトウェアの流通・流用が必須である．統合ノードシステムでは，共通プラットフォーム自体を各アプリケーションに流通する方法，クラスを部品として流用させる方法を採用した [2, 3]．

(1)　共通プラットフォームの流通

共通プラットフォームは，流通先での安定動作を保障しやすいロードモジュールで流通さ

せることとした．この際，ユーザによっては，あるリソースの容量をより多く必要とする，あるいはこの装置については不要であるというようなケースがある．よって，カスタマイズサポートデータ（CS データ）という概念を導入し，プログラム中の定数データ（リソース数最大値，装置搭載有無データなど）を，ツールを通して書き換えられるようにした．これによりユーザはプログラム自身には直接手を入れず，カスタマイズすることが可能となっている．

(2)　部品の流通

共通プラットフォームを利用するユーザは，自分自身で特有の装置管理プログラムなどを作成するケースもある．このような場合に対応するために，C++言語のインヘリタンスを用いた差分コーディングによる部品管理の枠組みを開発した（**図 4･10**）．装置管理プログラムにおいて，装置の冗長構成や特性に応じた基本部品（装置管理基本クラス）を 15 種類作成し，ユーザにおけるプログラムの生産性向上に役立っている．

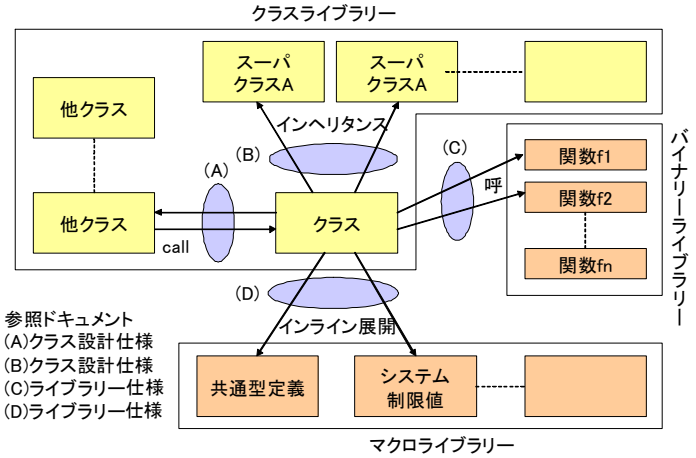


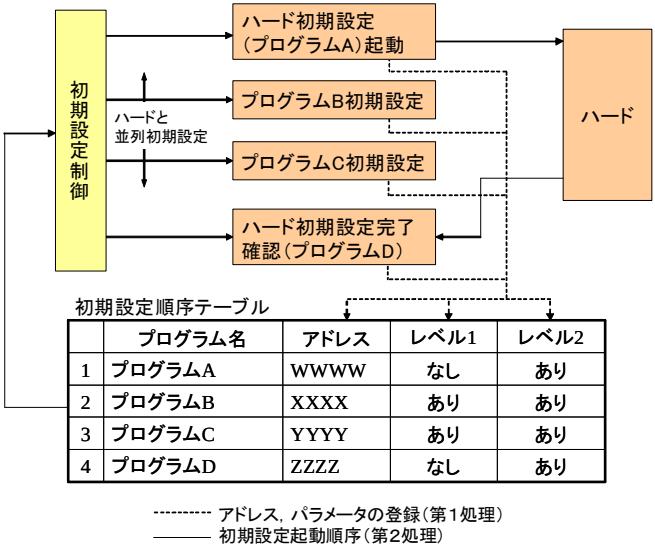
図 4・10　クラスの部品化方式

(3)　システムの差異を吸収する初期設定方式

初期設定制御は，搭載している装置が異なったり，ソフトウェアの初期設定順序が異なるなど，システムごとの差異がある．初期設定制御プログラムを各種通信システムで共通に使用できるように以下の構成を採用した（**図 4･11**）．

・装置管理に対する初期設定範囲の指定と呼処理に対する呼救済条件の指定は，レベル 1，2 などの論理化したパラメータで統一する．
・各処理プログラムの初期設定順序を記述したテーブルを設け，このテーブルの変更のみで流用を可能とする．
・初期設定順序を二つに分離する．第 1 の処理では，各処理プログラムが，初期設定起動アドレスと初期設定パラメータを初期設定順序テーブルに登録し，第 2 の処理では，初期設定順

序テーブルに従って各処理プログラムを起動する構成とすることで，初期設定制御プログラムが各処理プログラムの初期設定順序や初期設定パラメータを意識しないようにする．



	プログラム名	アドレス	レベル1	レベル2
1	プログラムA	WWWW	なし	あり
2	プログラムB	XXXX	あり	あり
3	プログラムC	YYYY	あり	あり
4	プログラムD	ZZZZ	なし	あり

図 4・11　初期設定処理の概要

このような初期設定順序テーブルの導入により，初期設定制御の汎用化が図れた．なお，初期設定起動処理と初期設定完了確認処理を設けて，ハードウェアの初期設定処理とハードウェアの初期設定処理を並列に行うことで，初期設定時間の短縮化も図った．

4-3-4 サービス無中断技術

(1)　ファイル更新技術

本プラットフォームでは，通話中呼を救済し，かつ呼の受付処理をできる限り中断せずに行うファイル更新技術を採用している．通話中呼を切断せずに新ファイルに切り替えるためには，すべての交換処理を中断・凍結して，呼データなどを新側プロセッサに引き継ぐ必要がある．このとき，新しい呼の受付処理が中断されるので，中断時間を可能な限り短くする必要がある．このために，現用／予備プロセッサを並列動作させ，呼データのみを中断期間中に転送し，処理中運用データ・所データを呼処理実行中に転送することによって，呼処理中断時間を短縮化した．

処理は以下のシーケンスにより行われる．

・新ファイルのローディングと新ファイルの初期設定
・運用データ，所データの新ファイル側への引継ぎ処理
・新ファイル側（元の予備系）への切り替え（呼処理中断）
・新側から旧側の呼データ読み取り
・呼処理の再開

(2) プロセッサ置換技術

ハードウェアの進歩や性能への要求に応えるため，統合ノードシステムでは，通話中呼を救済し，かつ呼の受付処理をできる限り中断せずにプロセッサ置換を行う技術を開発した [4]．この方式は，ファイル更新と同様の手順で実施できるなど保守性に優れ，また将来予想される新たなプロセッサに対しても，S1 インタフェース上位ソフトをほとんど変更せずに実現できる．

具体的には，ファイル更新手順の中で，予備系のプロセッサを新プロセッサに置き換えたうえで，新ファイルを起動し，運用データ，所データ，呼データを転送・形式変換することにより実現できる．

(3) プラグイン技術

ノードシステムの高信頼化の一環として，運用中のファイルを，提供サービスに影響することなく，部分変更する技術，「プラグイン」を開発した [5]．従来システムでは，運用中のプログラムの修正は機械語記述でかつ変更部分のアドレスを意識する必要があったので，実施上の困難さが大きく，また修正した機械語を原本のソースファイルにエディットし直す必要もあり，稼働的にも負担が大きかった．これらに対応するため，プログラマがソースレベル（C，C++）で修正，追加したプログラムを交換機内でオンラインリンケージできる機能を実現した．

プラグインの主要な機構は，交換機内で修正された新プログラムのエリア管理を行うこと，新プログラムから既存プログラムへの関数呼び出しやデータアクセスに関するアドレス解決を行うこと，被置換プログラムの先頭に新プログラムへのジャンプ文を埋め込むことである．必要なら新プログラムの初期設定を行うためのトリガをかけることも可能であり，新規プログラムの追加がしやすくなっている．

4-3-5 所データ生成技術

既存システムでの所ファイル作成においては，所データ作成システム（Data GeNeration：DGN）で機械語レベルの物理所データを作成した後，システムファイルとの結合という 2 段階の作業が必要であり，時間と稼働を要していた．統合ノードシステムでは，**図 4･12** のようにノードシステムと DGN 間のインタフェースを論理化し，この問題に対応した．

・DGN で各ノードの所データを作成する場合，所ごとに所データのメモリ割付を意識する必要がなくなったこと．
・ファイル更新作業においては，システムファイルと論理所データファイルの転送を個別に行うことが可能となったこと．

により，所データ設計・管理，ファイル更新時の作業の容易化・効率化を図った．

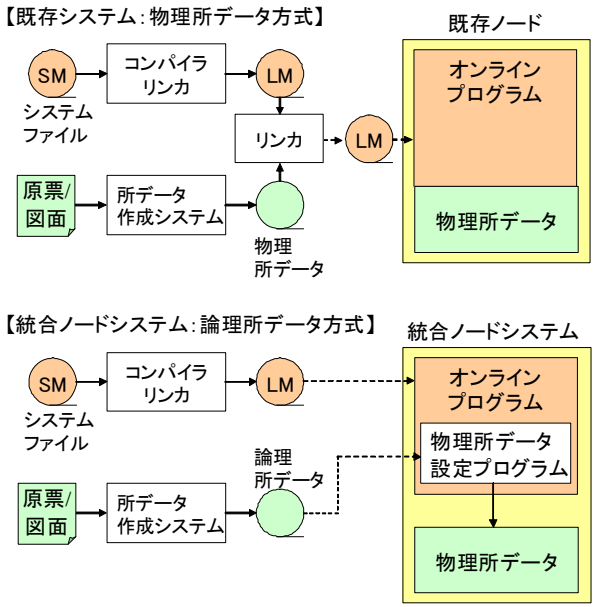


図 4・12　所データ生成方式

4-3-6 TMN 適用技術

TMN（Telecommunication Management Networks）では，オペレーションシステム～交換機（OPS-NE）間のコマンド・自律メッセージの送受信内容，転送プロトコル（CMIP : Common Management Information Protocol），インタフェース仕様記述（GDMO : Guidelines for the definition of MO（managed Object））などを標準化している．統合ノードシステムにおいて，これらを効率的に実現するため，

・TMN に従ったコマンド・自律メッセージ設計用上の共通ガイドラインの提供
・コマンド・自律メッセージ設計仕様書，内部形式仕様の自動生成技術の確立
・OpS へ送受する外部形式から，内部形式の変換プログラムの自動生成技術の確立
・GDMO にそった OpS-NE インタフェース仕様の自動生成技術の確立

を行った．これらにより，コマンドメッセージ設計に関したアプリケーションプログラムの開発の効率化，仕様の共通化を図ることができた．

具体的な TMN 機能の実現技術は**図 4･13** のとおりである．MO 管理部プログラムは，コマンド処理を行うプログラム（クラス群）と OpS との間でコマンド・自律メッセージを仲介するものである. MO 管理部は, 統合ノードシステムのどのシステムにおいても使用されるが, システムごとに必要となる保守機能に併せてチューニングする必要があり，これをサポートするツールとして MO 管理支援ツールを開発してコマンドメッセージ設計の効率化を図った [6].

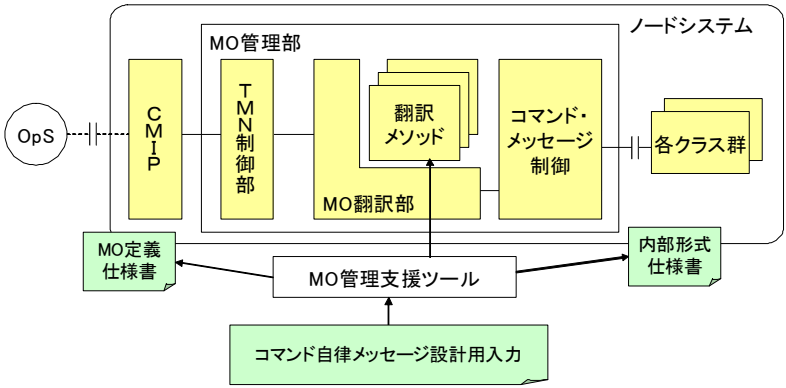


図 4・13　TMS 実現技術

（出典：NTT R&D 誌, vol.45, no.6, pp.49-56）

■参考文献

1) 小菊一三ほか, “ノードシステム用 IROS 特集, ” NTT R&D, vol.7, 1994.
2) T. Yamada, H. Sunaga, et al., “Object-Oriented Switching Software in C++ -Non-stop Service Enhanceable Software,” IEEE Globcom '94, 1994.
3) M. Furukawa, H. Sunaga, et al., “Analysis of Reusability of Communication Switching Software Based on C++ Object-Oriented Design,” IEEE ICC'96, 1996.
4) 二神　新, 中村宏之, 稲守久由, “階層化ソフトウェアにおけるサービス無中断プロセッサフィールドアップグレード技術,” 信学論(B), J83-B, no.10, pp.1410-1418, 2000.
5) H. Sunaga, et al., “Applicability Evaluation of Service Feature Enhancement using the Partial-File "Plug-in" Modification Technique,” IEEE ICC '96, 1996.
6) H. Inamori, K. Ueda, K. Koyanagi, H. Sunaga, and T. Kishi, “Applying TMN to a Distributed Communication Node System with Common Platform Software,” IEEE ICC '95, 1995.

■5群 － 4編 － 4章

4-4 ソフトウェア支援技術

（執筆者：村上龍郎）［2009年4月 受領］

ノードシステムのソフトウェア開発は，(1)大規模ソフトウェアの多人数による開発である，(2)ターゲットマシンと開発用マシンが違うクロス開発である，(3)ターゲットはリアルタイムシステムで超多重処理が要求される，(4) 24時間，365日ノンストップシステムである，という特徴があり，ツールとして市販品を組み込むものの，開発環境としてはノードソフトウェア開発環境として専用的に組み上げるのが一般である．ここでは交換機などのノードシステムのソフトウェアを開発するうえでの開発環境全般にわたって紹介する．

4-4-1 ソフトウェア開発用LAN

1980年代の終わり頃から大規模ソフトウェア開発では，それまでのメインフレームによるソフトウェア開発からネットワークを介したワークステーションによる分散開発にシフトしていった．ここで必須となったのが，開発用LANである．LANは多人数の大規模開発で次の役割をになった（**図 4･14** 参照）．

・ソースコードをはじめとするプロダクトの一元管理と，これらを使った並行作業を可能にした．
・エディット～コンパイル～LM化～実機へのローディングをLANを介して自動化することによって，作業の迅速化のみならず，ファイルの取り違いのような基本的なミスを撤廃する．
・設計～ファイル化はもとより試験作業に関しても，LANを介した居室での作業を多くし，作業の効率を高める．
・帳票ベースの情報通知を電子化し，ネットニュースやメールを用いて伝達・管理することにより，情報流通の徹底，多人数によるグループワークを支援する．

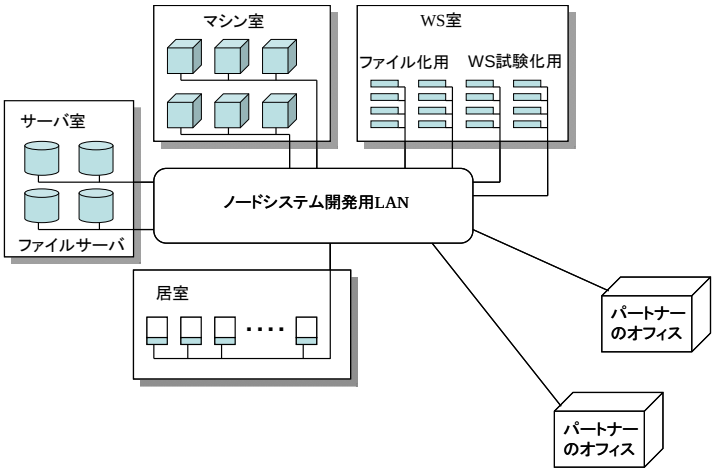


図 4･14　ノードシステム開発用 LAN の概要

4-4-2 作業標準及び作業マニュアル

ノードシステムの開発は多人数による大規模開発である．このため，作業標準，作業マニュアルは必須である．これなくして，マルチ（ソフトウェア）ベンダを前提としたノードシステムのソフトウェア開発はあり得ない．重要なことは，作業標準を徹底させることである．そのため，作業標準をドキュメント化して，開発用 LAN から即座に参照できることはもとより，各種ツールが作業標準をサポートしていることが極めて効果的である．特に，設計支援ツール，ファイル化サポートツール，帳票管理ツールには，それらの機能が埋め込まれる必要がある．また，各種ツールのマニュアル類を開発用 LAN から必要なときに即座に参照できる必要があり，オンライン化は必須である．

4-4-3 設計支援ツール

設計工程では様々な設計手法が提案されており，それに相当した市販ツール，独自開発ツールが使われている．ここでは，そこには触れず設計手法と独立なドキュメント作成ツールを紹介する．

●設計ドキュメント作成ツール

ノードシステムのドキュメント作成には次の条件を満足するシステムを採用している．

・UNIX ワークステーションを用いた開発．
・膨大なドキュメントの参照を容易にする，変更管理を容易にする観点からファイル単位を小規模（節単位など）にして，ハイパーリンク形式の構成で実現する．
・ペーパードキュメントとしての出力は従来の冊子形式をサポートする．

上記条件を満足する UNIX フリーソフトを活用したノードシステム用のドキュメントツールを開発し利用した．

4-4-4 ファイル化ツール

ファイル化は, ソースコード作成から実機用のロードモジュール作成までの工程である（**図 4･15** 参照）．ノードシステムのファイル化には以下の条件から物理的なオブジェクト（機械語）とプロセッサのアーキテクチャを強く意識する必要がある．

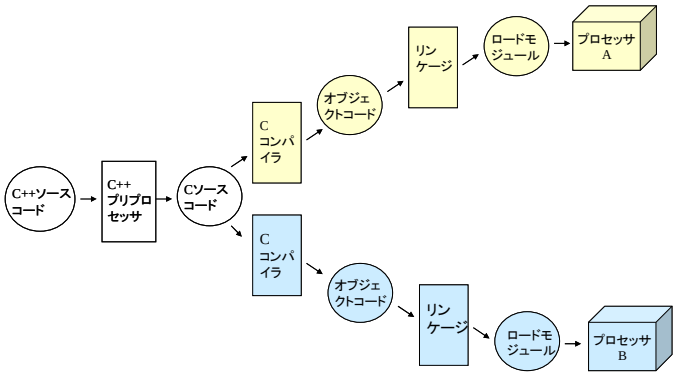


図 4･15　C++によるファイル化の流れ

・処理能力の要求が極めて高い．処理能力としてはダイナミックステップのみならずキャッシュのヒット率も大きく影響するため，機械語の展開，1 次メモリのマップをファイル化ツールによって最適化する必要がある．
・ノンストップ条件のため，再開処理には主にメモリクリア条件に段階的なフェーズを設けており，これに対応したメモリマップをサポートする必要がある．更には，近年のノードシステムは再開処理において，接続中の呼は初期設定しないという“レスキュー”が前提であるため，呼情報を別置きにして通常の再開処理ではクリアされないようなコードを作成する必要がある．
・ノードシステムでは，走行中にコードをアップデートする仕組み，オンラインパッチに相当する“プラグイン”が必須である．プラグインは，再開を伴わないオンラインパッチであるが，ファイル化という観点では，ソースコードレベルから作成する点，部分ファイル更新といえる．このため，プラグインのコード作成には，通常のファイル化ツールを使用するものの，吐き出すオブジェクトコード（ロードモジュール）には巧妙な工夫が必要となる．
・ノードシステムのファイル更新は，ノンストップ条件を満足する点，工夫が必要となる．基本的にはプロセッサの二重化（0 系と 1 系）とノンストップの再開処理を利用して行うが，新ファイルでメモリ割付条件の変更を吸収するという要求条件があり，これをファイル化ツールでサポートする必要がある．更に近年，ハードウェア技術の進展（特にプロセッサ）に伴いファイル更新がソフトウェアのバージョンアップのみならずハードウェアの更改も伴ういわゆる“プロセッサ置換”が条件の一つに加わった．プロセッサ置換は，ノードシステムのノンストップ条件を満たしつつ，ソースコードレベルでは同じものを使いつつ，全く異なったプロセッサのアーキテクチャに対応しなくてはいけないため，ソースコードの作成の制限条件やメモリ割付での工夫が必要となる．

以上の条件を満たすことを前提としたファイル化ツールを紹介する．ここでは，近年，ソフトウェアの生産性向上から採用されたオブジェクト指向によるソフトウェア開発を前提に C++での開発を想定したファイル化ツールの例を示す．

(1) C コンパイラ

C コンパイラは基本的にターゲットシステム（プロセッサ）に対応したいわゆる“ネイティブ”のコンパイラを利用する．

(2) C++プリプロセッサ

マルチプロセッサやプロセッサ置換を意識して，コンパイラはネイティブな C コンパイラを使用すること，プラグインやプロセッサ置換でソースコードに制約があること，オブジェクト指向設計による生産性向上を狙う，という三つの条件を満足する上で C++プリプロセッサは極めて重要な役割を果たす．また，ノードシステム用のターゲットマシン（プロセッサ）に依存しない C++プリプロセッサの開発が必要となった．

(3) リンケージツール

リンケージツールもプロセッサ対応のネイティブのリンケージツールを利用する．ただ，

割付条件作成に関しては前述したように，(a)処理能力向上，(b)様々なフェーズに対応した再開処理の実現，(c)プラグインの実現，(d)プロセッサ置換の実現，のためにメモリ割付は極めて重要となる．これを容易に実現するためのツール（メモリマップサポートツール）の開発が必要である．

(4) コーディングチェッカ

前述したようにオブジェクトコードをリロケータブルに置き換えるプラグイン，プロセッサ置換を考慮してプロセッサのアーキテクチャに極力依存しないソースコードの作成には，C++のコーディングに制約条件が必要である．更に，処理能力を増すコーディング，オブジェクト指向のポリシーを徹底して，今後の機能追加をスムースに行うためのコーディングガイドラインがある．前述した C++プリプロセッサと C コンパイラでは，C++記述のシンタックスエラー，セマンティックエラーの抽出はサポートするものの，上記，コーディングの制約やガイドラインをサポートするうえで，コーディングチェッカは生産性向上に重要な役割を果たしている．

(5) プラグイン作成ツール

(a)オブジェクト指向のメリットを生かしたオブジェクトレベルのファイルの差し替え，(b)ノードシステム特有のノンストップオンラインパッチのサポートの要求条件からプラグインの存在は，現在のノード開発に必須である．

プラグインの基本的メカニズムは，オブジェクト単位に更新ファイルをプラグイン専用エリアにロードして，オリジナルなオブジェクトのロードモジュールからリンケージを張るというものである（**図 4･16** 参照）．

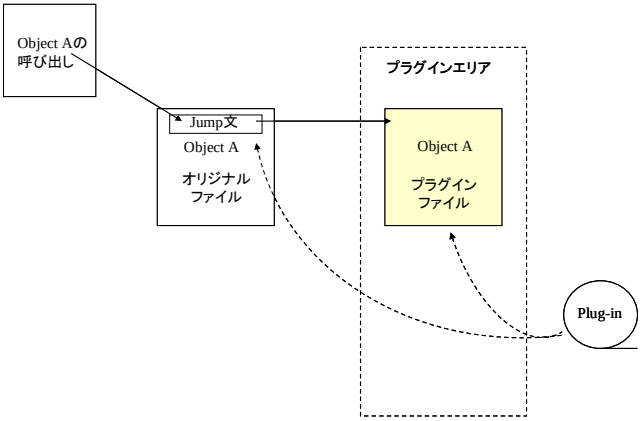


図 4・16　プラグインのメカニズム

プラグイン作成に必要な条件，オブジェクトのリロケータブル性，(スタックを含む) テンポラルデータの同一性は，コーディングチェッカや C++プリプロセッサでサポートされるものの，プラグインエリアへのオブジェクト配置とオリジナルコードからそこへのリンケージ

は，C++プリプロセッサ，C コンパイラ，（メモリマップサポートを含む）リンケージツールと連携して動作するプラグインサポートツールの作成が必要となる．

4-4-5 試験ツール

ノードシステム開発で試験工程の占める割合は多大であり，開発稼働の多くをここで費やす．そのため，開発環境も試験工程，特に実機試験には多くの工夫がなされている．実機試験では，擬似呼試験機，負荷試験機，オンラインモニタなどのハードウェアツールが活躍する．ただし，これらのツールはターゲットシステム及びターゲットサービスに強く依存するため，ここでは割愛して，試験工程の生産性向上に寄与するシンボリックデバッガ，シミュレータ試験ツールを紹介する．

試験工程は大雑把にいって，単体試験，結合試験，複複試験・安定化試験，に大別される（**図 4･17** 参照）．

- **単体試験**：（オブジェクトなど）ソフトウェア部品のスペックを試験する工程である．試験の方法としては，ソフトウェア部品のイン／アウト条件を規定して，これが満足するかを試験する．
- **結合試験**：ソフトウェア部品を結合した機能レベルの試験である．“機能試験”と呼ばれることもある．試験方法として，ソフトウェア機能としてもたらす条件（イン／アウト条件など）をチェックするシステム機能試験と，擬似呼などを用いたエンドーエンドの試験を行うサービス機能試験がある．
- **複複試験・安定化試験**：複複試験は，複数複合試験の略称である，ノードシステムの特徴の一つとして，超多重処理がある．そのため，超多重処理がうまくいくか，サービス及び複数呼の競合処理がうまくいくかをこの工程で試験する．

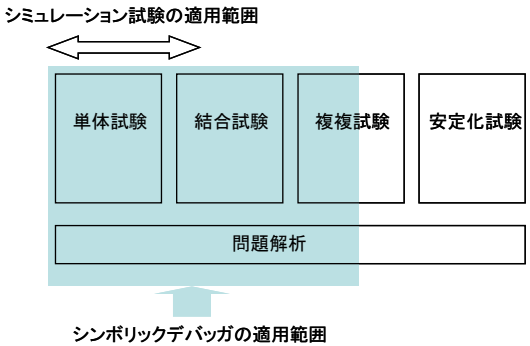


図 4・17　試験工程と各ツール

また，安定化試験には，ノンストップ性を確認する“長時間安定化試験”，処理能力をチェックする“負荷試験”，システムの過酷状況を作り出し，それに対するロバストネスを確認するいわゆる“いじめ試験”，また，ターゲットシステムによっては実際のネットワークを模擬したネットワークを組んで，システム単体のみならずネットワークレベルで正しい動作をするかを確認する“ネットワーク試験”が含まれる．

上記の試験の各工程と並行して行われるのが“問題解析”である．問題解析は，各試験工程で問題が見つかった場合に，問題を再現してバグの所在を見つけ出すことと，バグを修復した場合にそれが正しく修復されているかを確認する作業が含まれている．また，バグ修復に伴っては，ソースコード変更の影響を確かめるため，単体試験，結合試験，一部複複試験をやり直すリグレッション試験が行われる．

(1) シミュレーション試験ツール

ノードシステムは原則的にクロスターゲット（実機と開発環境が異なる）であるが，シミュレーション試験は実機を使わずに開発環境（主にワークステーション上）にシミュレータを用意して行う方法である．コストの高い実機の試験用に使う台数を削減する，実機を使わずにワークステーションに閉じた試験によって，試験のターンラウンドタイムの削減や（実機の操作を必要としないことによる）試験への参加人数を削減して（基本的にシミュレーション試験は 1 人）試験稼働を削減するなどの生産性向上を目的としている．シミュレーション試験は，単体試験の全工程，結合試験の一部（主にシステム機能試験）に利用される．

シミュレータには，ソースレベルでソフトウェアの実行をなぞる“ソースレベルインタープリタ”とオブジェクトコードをワークステーション上で実行する“実機シミュレータ”がある．前者は，言語（例えば C++とか C とか）対応に用意すればよいのに対して，後者はプロセッサ対応，ある意味ではコンパイラ対応に準備する必要がある．ただ，試験工程の目的から考えると，前者は単体試験の一部に使用する基本はコーディングレベルのサポートツールであるのに対して，後者が試験工程のシミュレータの役割を果たしており，実機シミュレータの準備がノードシステム開発では重要となる．

(2) シンボリックデバッガ

前述したように，ノードシステム開発では，プロセッサのアーキテクチャに対応した機械語やメモリ割付が重要な役割を果たし，この条件なしではノードシステム開発はなし得ない，試験工程でも同様である．

しかしながら，試験工程はソフトウェア開発の全工程で最も稼働をかける工程であり，機械語やプロセッサのアーキテクチャを理解しているエンジニアを多く集めることは不可能に近い，また，オブジェクト指向を利用した設計手法を採用しても最も肝心の試験工程でオブジェクトコードを意識したのでは，オブジェクト指向設計のメリットが大幅に減ってしまう．これらの意味からノードシステム開発にもシンボリックデバッガは必須といえる．

シンボリックデバッガは，主に単体試験，結合試験，複複試験の一部に適用されるが，特にそれらの工程の問題解析でその威力が発揮される．なお，安定化試験（複複試験の一部も）は最適化コンパイルを適用して実際の運用と同じ条件で試験を行うためシンボリックデバッガの利用が難しくなる．

シンボリックデバッガは，プロセッサ使用，すなわちネイティブコンパイラと強く連携するセット商品ではあるが，ノードシステム特有の割付条件の反映や試験実施におけるマンマシンインタフェースの統一を行うためにプロセッサ依存の市販品を利用しつつも，ノードシステム用としてのカスタマイズが必要となる．

4-4-6 帳票管理ツール

従来，紙ベースの帳票を電子化したものであるが，ノードシステムの多人数による大規模開発を効率よく実施するための連絡手段及び管理手段として重要な役割を果たす．特に以下の管理システムとは深く連携することが必須条件である．

- **問題処理管理**：問題の発生から解決までを管理するシステムである．問題の解決を促進するばかりでなく，ノードシステムの大規模ソフトウェア開発では，問題の発見者，解析者，修復者が異なることから，それらの間で情報を共有すると意味からも重要な役割をになう．
- **バグ管理**：いわゆるソフトウェアの品質管理である．どこの部分にどのようなバグが何件あったかをリアルタイムに管理するシステムである．そのため個々のバグ情報を格納するだけでなくリアルタイムでの集計機能が必須となる．そもそも試験工程の目的は品質確保，バグを見つけ出し，修復することが目的でこれをいかに徹底してかつ効率良く叩き出すかが試験工程でのミソである．そのため，バグの発生状況を随時把握しながらの試験実施計画が必要となる，具体的にはバグ発生曲線（予想バグ数に対する抽出度合いと試験項目に対するバグの発見件数から把握するバグの枯れ具合）に基づいた試験の各工程の移行判定，試験終了判定にこの管理システムの存在は必須である．
- **ソースコード管理**：ソフトウェア開発の基本である．しかも，ノードシステム開発のような大規模・多人数開発では，これが整っていなければ，様々なトラブルを引き起こし，そもそも試験や問題解析が何をやっているかわからないという根本をゆるがす事態となる．ソースコード管理の要件は，ファイル部品単位にバージョン管理を行うことである．バージョン管理には，ソースコードの変更理由，バグの修復との関連，試験を行う（行った）ロードモジュールとの関係付けを電子的にサポートし，何に対する試験が行われたか，修正部分が確実に確認されたか，ファイルの上書きや取り違いによるデグレードがないかをシステム的に保証するものである．

帳票管理ツールは，上記情報管理システムと深く連携して，情報管理システムが大規模・多人数ソフト開発の日々の作業の中で機能するためのいわゆるグループウェアツールの位置づけである．

(1) 問題処理票

試験時の問題の発生から，解析状況，バグの所在，修復確認までのそれぞれの役割をする異なる担当者間に伝える連絡手段と同時に問題のライフタイムでの履歴をトラックできるための手段である．そのため，そのフォーマットには，連絡用としての必要事項が盛り込まれていること，適切な担当者や管理者に伝わるための機能をサポートしていること（開発用 LAN を用いたメールシステムとメーリングリストの活用），問題処理管理と深く連携しており，一つの問題に着目した最新の状況が帳票を索引することによって把握できることが要求条件である．また，検索機能，一覧表示機能，集計機能が具備され，問題単位，ソフトウェア部品（機能ブロックやオブジェクト）単位，担当者単位での検索ができる，未解決問題が即時に表示できる，バグの集計が容易になるなどの要件が満たされている．なお，バグの集計には後述する変更票との連携によってなされる．

問題処理票は，以下のタイミングでそれぞれの担当者によって起票されるが，それぞれがリンクされることによって，発生問題ごとにあたかも一つの帳票として管理される（**図 4･18**

参照)．

・問題の発見：発見者（試験担当）
・問題の解析：解析者（機能部品担当）
・問題の解析：解析者…これは問題の原因究明まで被疑の機能部品担当によって繰り返される
・問題の修復：ソフトウェアバグの場合は「変更連絡票」よってなされる
・問題修復の確認：一般的には発見者（試験担当）

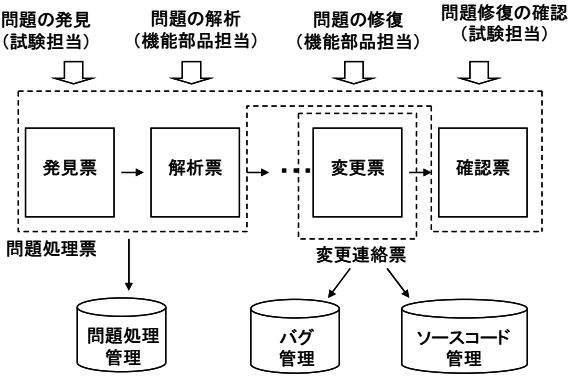


図 4・18　各帳票と管理システム

(2)　変更連絡票

ソースコードの変更に伴い，変更箇所や変更理由を記入するものである．ソースコード管理ツールと強く連携する必要があり，そのバージョン／リビジョン管理とリンクして，変更箇所が変更差分として抽出できる．逆にソースコードのバーション／リビジョンアップした理由は，変更連絡票とのリンクでソースコードから容易に参照できる．

この帳票のフォーマットには，変更理由として，機能追加／バグ修正を明記して，機能追加の場合は，どの機能（サービス）に属すか，バグの場合は，バグの詳細情報（自責／他責），バグの混入工程，バクの発生理由など）を記述するファーマットを具備する．これによって，機能追加ごとの開発規模の集計を容易にする，バグの集計や詳細分析を容易にすることができる．