無線回線制御技術の進化と深化

Evolution and deepening of the wireless communication control technology

磯部 洋次*
Yoji Isobe

無線通信分野は、＜第1世代（1G）＞アナログ方式の携帯電話（FDMA方式）、＜第2世代（2G）＞デジタル方式携帯電話（TDMA方式）、＜第3世代（3G）＞デジタル方式携帯電話（W-CDMA方式）と無線周波数の有効利用に向けたデジタル化が進むと共に、インターネットの普及によるパケット伝送の需要が急増、低コスト化・高速化・低遅延化のための伝送技術の標準化、さらに次世代への大容量高速通信化へと変化を遂げている。

無線回線制御技術においても、制御するデータ量ならびにユーザ数（呼数）が増大し、リアルタイム組込ソフトウェアの重要性が高まる中、当社は、通信制御装置の組込ソフトウェア開発に従事し、時代に応じた市場の発展に伴うキャリアの要求に応えるべく、経験を積んできた。

本論文では、これまでの開発経験を踏まえた無線回線制御におけるソフトウェア設計に関する変遷（構造化設計からオブジェクト指向へ）、ならびに、アーキテクチャの進化と深化について述べる。

In the field of wireless communication, with the advance of the digitization to utilize the radio frequency, throughout ＜the first generation（1G）＞ analog cell-phone（FDMA method）, ＜the second generation（2G）＞ digital cell-phone（TDMA method）, and ＜the third generation（3G）＞ digital cell-phone（W-CDMA method）, the technology has changed to rapidly increase of demand for packet transmission by the spread of Internet, the standardization of the transmission technology for price reduction, speeding up and low delay, furthermore a large-capacity high speed communication toward the next generation.

In the wireless communication control technology, quantity of to control data and the number of users（the number of calls）increased, and importance of real-time embedded software is increased.

Under such situations, we have engaged in development of real-time embedded software, and acquired experiences to meet the demand of the carriers with the development of the market.

In this article, we describe the change about the software design（from a structured design to object-oriented design）on the basis of past development experience and evolution and deepening of the architecture in the wireless communication control.

1. まえがき

1980年代に登場したアナログ方式は音声中心の携帯電話であったが、1990年代にアナログ方式からデジタル方式へ、加えて、低速データ通信も始まった。2000年代後半には、音声、画像、高速データ通信が広まり、公衆網のIP化に伴う大容量高速通信の需要が高まり、現在では、第四世代（4G）に向けた拡張が行われており、当社が無線通信制御装置（無線回線制御技術）の開発に携わったのは1993年頃で､90年代には第二世代無線基地局、無線制御装置、第三世代無線基地局のS/W開発を経て現在に至る（図１）。

2. システム概要

一般的に無線通信インフラのシステム構成は、図２のようになる。

伝送装置は、ネットワークを通じて、情報（データや音声等）を伝える装置である。例えば、電話のように有線や無線で情報を伝達する際、外部からの雑音などの影響により、情報が欠損したり、破損して正しい情報が伝

＊関西事業部 第三技術部

わらなかったりすることがあるが、伝送装置で情報を伝達しやすい信号に変換することで、相手に正確に情報を伝えることができる。

基地局は、移動局との通信を行うための装置である。有線網と無線網の末端に位置し、有線・無線間の情報変換、移動局の管理、制御を行う。

移動局は、携帯電話、船舶・航空機の局のように、移動する無線局のことである。近くの基地局と通信を行い、情報を発着信する。

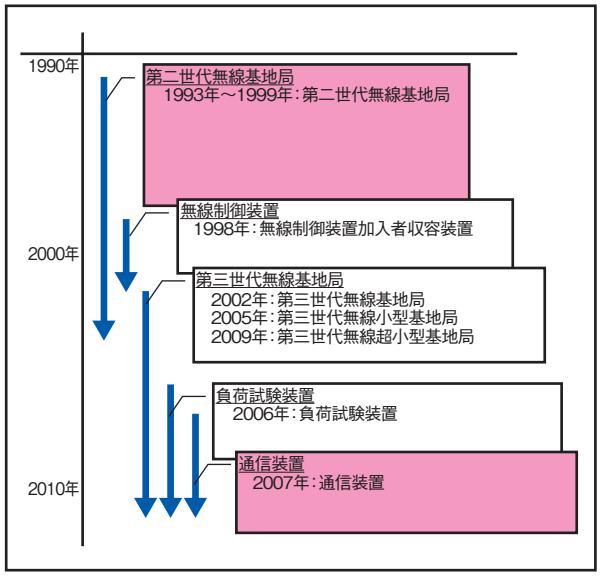

図１　当社の歩み

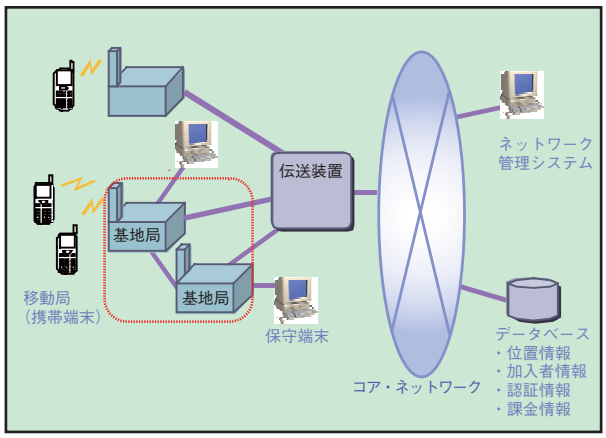

図２　システム構成図

3. ソフトウェア・アーキテクチャ

無線通信制御装置では、大きくは４つの機能ブロックで構成される。①無線管理、②認証管理、③呼制御、④運用・保守管理である。

90年代の無線通信制御装置のソフトウェアのアーキテクチャでは、仕様変更・追加に強い設計が求められ、４つの機能ブロックをさらに機能単位で分割するような構造化設計が主流であった。

一方、最近の開発では、性能や信頼性だけでなく、仕様変更・追加の容易性ならびに短期開発も重視されるようになってきており、従来の構造化設計からオブジェクト指向設計に変遷を遂げている。

以降の章では、1990年代に開発したA装置と2000年代に開発したB装置を例に、ソフトウェア・アーキテクチャの変遷もしくは変化について述べる。

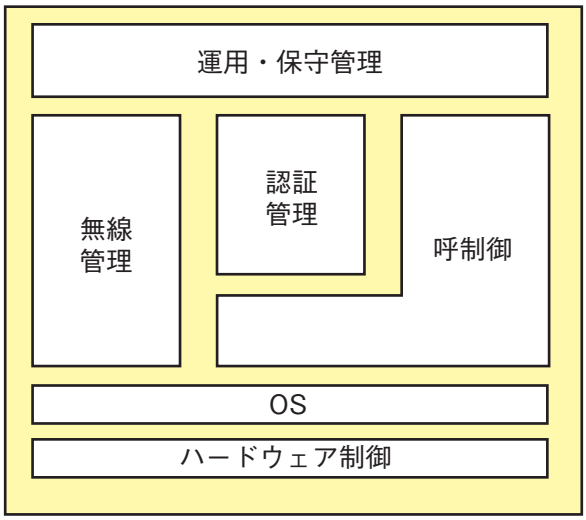

図３　機能ブロック図

3.1　A装置

A装置は、１つの装置がカバーするエリア範囲が狭く、収容数も少ない小規模システムである。
アーキテクチャ設計では、過去の実績（流用）も視野に、構造化設計を採用した。

3.1.1　ソフトウェア構成

A装置のソフトウェア構成は、当該システムが並列処理する機能単位でタスク分割し、リアルタイム性を確保する構造とした。

例えば、無線チャネルの割当、閉塞制御、リスタート処理、規制制御、再同期制御等、チャネル全般に関連する機能はチャネル管理タスク、同期確立、電解監視、通信品質監視、同期はずれ制御、チャネル切替、切断、エア側（L2以下）障害監視、通信品質報告等の通信チャネルに関わる機能はリソース管理タスクというように、機能とタスクを紐づけた構成となっている。

本装置のソフトウェア構造のように、機能単位でタスクを分割することのメリットは、機能構造が明確でわかりやすく、設計誤りが見つけやすい点にあるが、システ

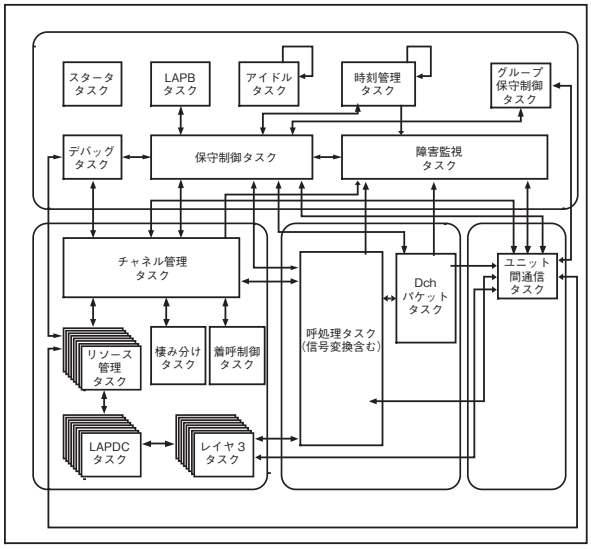

図４　A装置ソフトウェア構成図

ムが複雑になると、構造化設計での開発は処理を複雑にしてしまう可能性があり、かつサービス追加に伴う仕様変更では、影響範囲が多岐に渡り、デグレードの誘発、開発工数・工期の肥大化等のデメリットを経験した。

3.2　B装置

B装置では、「高信頼性」「高性能」の実現要件が条件であり、かつ大規模システムであるため、アーキテクチャ設計においては、従来の構造化設計からオブジェクト指向設計とした。ただし、本装置のようなイベントドリブン方式のS/Wでは、オブジェクトは受動的となり、複数イベントが発生した際に時間制約条件が異なるものを限られたCPU能力で、すべて満足できるかどうかを立証することが難しくなる。そこで、アクティブオブジェクトの概念に基づきタスク設計を行った。

また、アーキテクチャ設計を行うにあたり、システムスペック上の観点に加え、開発環境（大規模、短納期）、仕様変更のし易さを考慮して、呼処理の「通信基盤フレームワーク」を構築した。

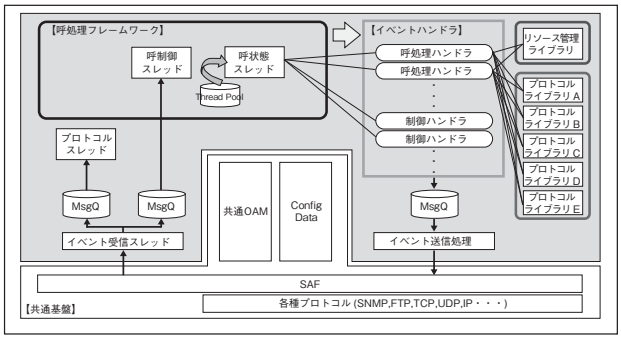

図５　アーキテクチャ構成図

3.3　通信基盤フレームワーク

通信基盤フレームワークのアーキテクチャは、RUPの4+1ビュー（ユースケースビュー、配置ビュー、プロセスビュー、論理ビュー、実装ビュー）の考え方に基づき、規定・設計した。

3.3.1　ユースケースビュー

当該ソフトウェアの機能網羅性を観点に、適用するシーケンスのパターンを抽出、それらを全て処理できるアーキテクチャの実現により、設計の妥当性を担保した。

3.3.2　配置ビュー

配置ビューについては、スコープを呼制御のみとしたため、ソフトウェアと実行環境の対応が1:1となり、今回の設計では対象外とした。

3.3.3　プロセスビュー

呼制御プロセスのソフトウェアは１プロセスで動作し、並列処理が必要なクラスについては、スレッドで実装する。スレッドで動作するクラスは、論理ビューで示した論理クラスのうち、アクティブオブジェクトとして規定されているもの（各パッケージの論理クラス図内で

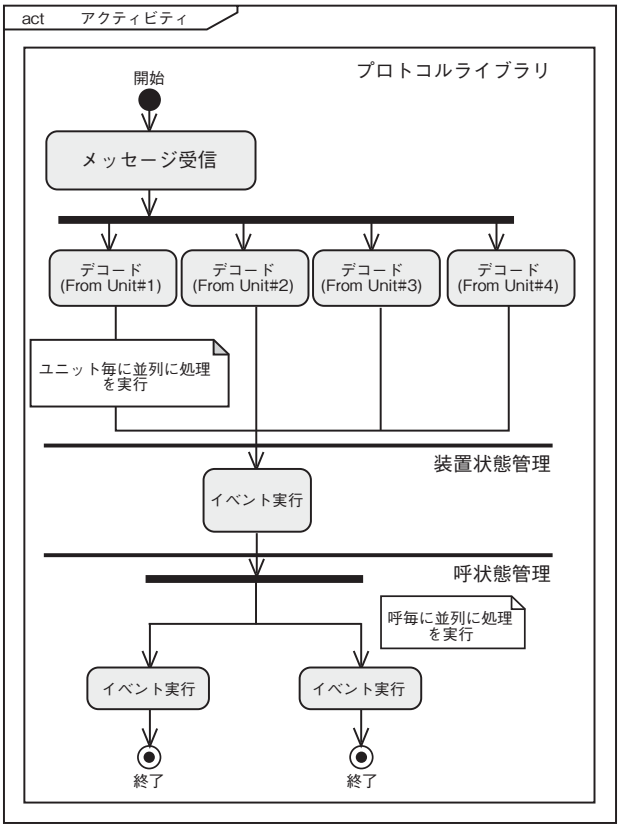

図６　並行性設計のアクティビティ図

スレッドと明記したもの）が該当する。スレッドの優先度については、応答時間の制限よりも、スループットの重要性を加味し、タイムシェアリングでOSが動的に割り当てることとした。並行処理を実施する箇所を図６に示す。

3.3.4　論理ビュー

機能分割した各機能は論理パッケージとして表現し、論理パッケージ内にはパッケージの機能を構成する主要な論理クラスを規定、また、規定した論理クラス群が、ユースケースビューで抽出したユースケースをどのように実現するかの検証を行った。全パッケージに共通した基本方針は以下のとおりである。

⑴　論理的なレイヤ構造として、仕様に特化するApplication Layerと、OSのシステムコール等、仕様に特化しないSystem Layerの2つに分けた。このレイヤ構造により、実行環境に依存する処理は、System Layerで吸収できる。また、ハードウェアの性能に応じたチューニング、実行環境の不具合等の影響範囲の絞り込みが容易になる。

⑵　CPUは効率を最大限に活かすため、アーキテクチャが複雑にならない範囲（保守性とのトレードオフ）でスレッド化した。なお、スレッドは、論理クラス図でアクティブオブジェクトとして定義する。

⑶　Application Layer内のパッケージ分割については、各パッケージが明確な責務を持ち、疎結合となるようにした。このことにより、保守性の向上を図っている。

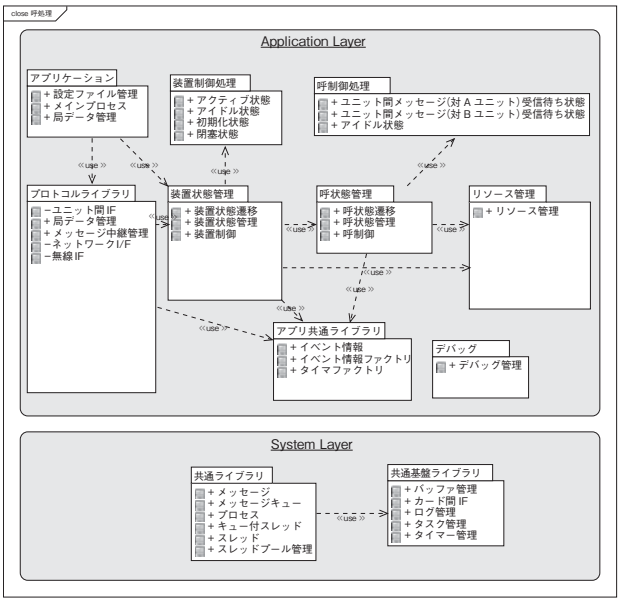

図７　論理パッケージ構成図

⑷　メモリの動的確保は行わない。インスタンスはアプリケーション起動時に必要数生成しておき、インスタンスが必要な場合は、Factoryクラスから払い出す。不要になった場合はFactoryクラスへ返却する。

⑸　スレッド間通信では、インスタンスのコピーは行わない。ポインタの受け渡しとする。

論理パッケージ構成は、以下の通り。

アプリケーションパッケージ：

メインパッケージであり、アプリケーション全体を制御するクラスから構成する。独自のコンフィグレーションファイルの管理や局データへのアクセサクラスを提供する。

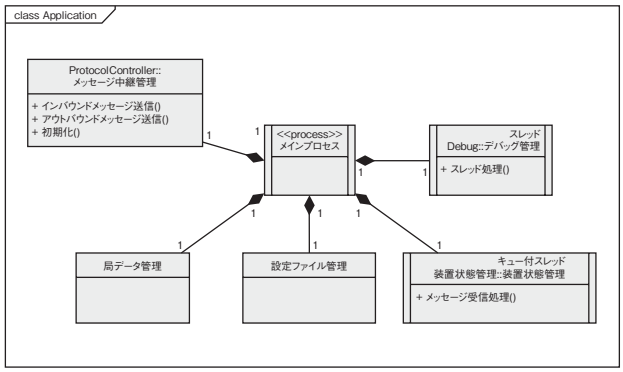

図８　アプリケーションパッケージ論理クラス図

デバッグパッケージ：

本ソフトウェアの開発者向けのデバッグ機能を制御するクラスから構成する。

アプリ共通ライブラリパッケージ：

Application Layerのパッケージにおいて共通で使用するユーティリティ的なクラス群から構成する。タイマの管理を行うクラスや、受信したイベントの処理に必要なデータを保持するセッションクラスが本パッケージに含まれる。

プロトコルライブラリパッケージ：

他ユニット間との通信、エンコード/デコード等、プロトコル処理に特化したクラスから構成する。他パッケージは本パッケージを介して他ユニットとの通信を行う。呼を意識しないプロトコル毎に汎用的な処理の責任を持つ。プロトコルのレイヤ、コネクション毎に並行して処理を行うことで、処理性能の向上を図る。

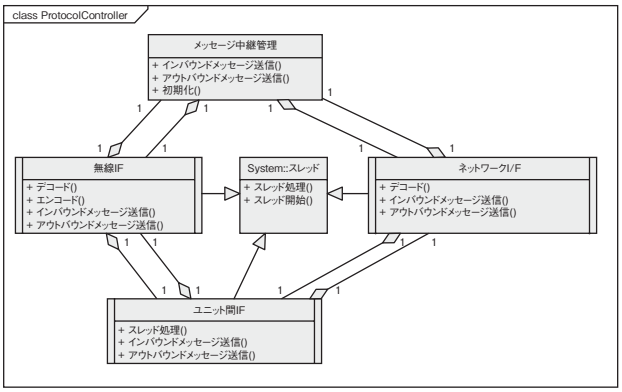

図9　プロトコルライブラリパッケージ論理クラス図

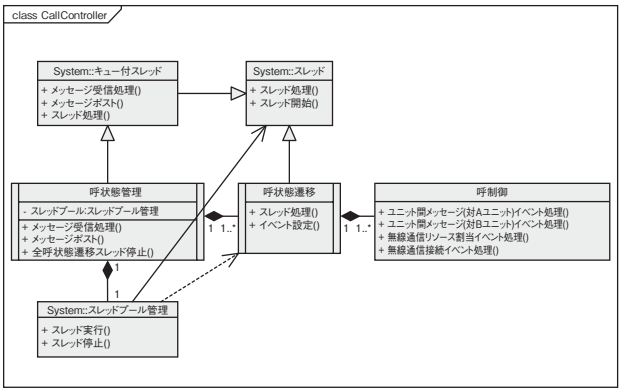

図11　呼状態管理パッケージ論理クラス図

装置状態管理パッケージ：

ユニット状態および運用・保守系のイベント処理を管理するクラスから構成する。全ての呼の一括処理、呼を意識しない処理の責任を持つ。

プロトコルライブラリからのメッセージは、本パッケージで終端するか、呼毎の処理であれば呼状態制御パッケージに処理を委譲する。

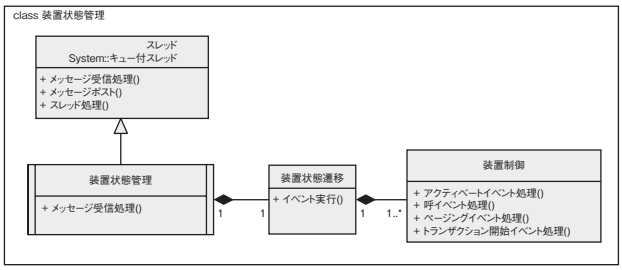

図10　装置状態管理パッケージ論理クラス図

状態制御処理パッケージ：

状態に応じたイベント処理を実行するクラス群から構成する。StateパターンにおけるConcreteStateに該当し、本パッケージのクラスは装置制御クラスから派生する。

ユニットの状態遷移における各状態をクラスとし、イベントはクラスのメソッドとなる。

呼状態管理パッケージ：

呼の状態を管理するクラスから構成する。特定の呼を対象とした処理の責任を持つ。呼毎の処理を並列化し、処理性能の効率化を図る。

受信したイベントから呼を識別した上で、処理を実行する。当該オブジェクトが存在しない場合は新たにオブジェクトを生成し、呼が解放された時点でオブジェクトを削除する。

また、オブジェクトを起動時に予め最大数プールしておき、オブジェクトの動的生成に要する処理時間のオーバーヘッドを回避した。

なお、呼状態管理クラスは特定の呼ごとの状態管理、および状態毎の処理を呼状態遷移オブジェクトとしてスレッドプール管理クラスに渡すことで、当該処理の実行を委譲する。

呼制御処理パッケージ

呼としての状態に応じたイベント処理を実行するクラス群から構成する。

本パッケージのクラスは、ひとつの呼の状態遷移での各状態をクラスとし、イベントはクラスのメソッドとなる。

リソース管理パッケージ：

通信リソースを扱うクラス群から構成し、通信リソースは、本パッケージで一元管理する。

本パッケージのクラスは、通信を実現するために必要な無線区間、有線区間、他ユニット間の物理/論理回線を制御するための資源を一元管理するクラスである。通信路を生成する際、関連するリソース管理オブジェクト種別毎にインスタンスを生成し、各種インスタンスを相互管理することで論理的に通信回線の制御を行う。

共通基盤ライブラリパッケージ：

共通ソフトウェアをラップするクラスから構成する。C言語APIで提供される共通ソフトウェアからさらに高水準のC++APIを提供する。

共通ライブラリパッケージ：

仕様に特化しないスレッドやメッセージ制御、キュー制御等の汎用的な機能を提供するクラスから構成する。システムコールを隠蔽し、高水準のAPIを提供することを目的とする。

3.3.5　実装ビュー

本アーキテクチャにおける機能面の検証はプロセスビューのユースケース実現で行ったが、性能面等の動的な検証については、プロトタイプを作成して行った。

4. むすび

近年の通信制御装置のソフトウェア開発は、「大規模化」「高信頼性」「高性能」要件に加え、「短工期」、「低コスト化」が求められ、また、市場投入後のサービス追加による仕様追加・変更の容易性が求められ、加えて、短工期、大人数での同時並行開発に耐え得るアーキテクチャが必要になる。さらに、他社との競争力強化の観点でも機会損失を防ぐ手立てが必要となり、これら課題を解決するには、

- レガシーな構造化設計からオブジェクト指向設計ソフトウェアへの変革
- 基盤フレームワーク適用による「方式設計の流用率向上」

の実現が不可欠であると考える。

本論文で紹介した通信基盤フレームワークも、イベント処理の並列化、状態遷移・シーケンス制御のフレーム化、システム固有のビジネスロジックを分離しており、上述の課題解決の実現に向け、一歩近づいたものと考える。

今後は、この通信基盤フレームワークを洗練化させ、他機種、他装置への横展開を推進し、要件から実装までのトレーサビリティ確保、ならびに、開発手法をガイドライン化し、ソフトウェア開発における「包括フレームワーク」として、深化させていく所存である。