次世代ネットワークサービスを支えるリアルタイムOSSへの考察 ――テレコム分野で活用がすすむ次世代OSS基盤

Realtime OSS Platform supporting Next Generation-Network Services ――Introduction for NGOSS Platform leveraged in Telecom Industry

井　上　　純

要　約　インターネットの爆発的普及に伴うネットワーク帯域需要の急激な拡大により，電気通信事業者のサービスは大きなパラダイムシフトを迎えている．これまで，通信サービスを支えるOSS（Operations Support Systems）は，可用性，信頼性，および，パフォーマンスといった，具体的な要件を満たすために，独自のハードウェア装置やプラットフォームの上に構築されてきた．結果として，オープン化や標準化の促進を阻害し，旧態然としたアーキテクチャは，市場の急激な変化に柔軟に対応することができない状況を作り出している．

このように，大きな変換期を迎えたOSSでは，従来にない堅牢でパフォーマンスに優れた新しいオープンプラットフォームを求めており，いくつかの業界コンソーシアムやコミュニティでは具体的な仕様検討や製品実装がはじまっている．

本稿では，OSSプラットフォームの抱える課題を取り上げ，理想的なアーキテクチャを考察すると共に具体的なソリューション例を紹介する．

Abstract　The Internet explosion has brought the broadband age forcing telecommunications carriers face the largest paradigm shift that they have never experienced.

In order to support the specific requirements such as the availability, the reliability, and the performance in terms of OSS(stands for" Operations Support Systems "), they had no choice but to implement their systems with using the special hardware or platform.

The legacy architecture makes the system difficult to keep up with the current rapid market changes. Thus, OSS requires an open but robust and high performance software platform that leverages the next-generation network services, so that some consortiums and communities have been challenging to provide concrete solutions.

This paper is to approach the problem statements that the current OSS is being faced, and is to consider what the ideal " to-be-model " as the next generation OSS platform will be.

1. はじめに

e-Japan戦略により，インフラの整備が当初の予想を上回る速さで進展し，日本では，世界でもっとも低廉かつ高速なブロードバンド環境が実現している．電気通信事業者においては，通信サービスにおける収益構造が大きく変化してきており，2010年のユビキタスネット社会（u-Japan）の実現に向けて，高度な通信サービスや情報システムの整備が必要になってきている[1]．このように，次世代ネットワークの構築や研究開発が着々と進む一方，電気通信事業者が通信サービスを運営するための基幹業務システムであるOSS（Operations Support Systems）分野では，このようなパラダイムシフトに備えた具体的なソリューションが次々に世に送り出されているが，その実行基盤として以下の要件を満たす実践的なソフトウェアプラッ

トフォームは数少ない．

第一の課題は，帯域需要の拡大に伴うネットワークを通した音楽や動画配信など，提供するサービスが益々多様化する中で，OSS が取り扱わなければならないシステムデータ量が急激に膨張している点である．より付加価値の高いサービスをタイムリーに提供するために，これまで以上にリアルタイム処理への要求が高まってきている．

例えば，ネットワークを通した音楽や動画配信では，大容量コンテンツのブロードキャストを可能にするバックエンドシステム，Pay per View や従量制課金など，きめ細かな課金方式への対応，コンテンツ著作権保護やセキュリティ要件への対応が不可欠であり，OSS に求められる処理能力とリアルタイム性への要件は帯域需要に比例して急激に高まりつつある．2000 年当時に総務省をはじめ，多くの有識者によって予測されていた「100×」通信社会は既に到来しており，これまでは単純にサービスネットワークの制御・管理が主な機能であった OSS はその要件が急変してきた．OSS が取り扱わなければならないデータ量は，従来の電話網に比較してここ数年で 2 桁以上増加したと言えよう．

一方，従来の OSS は，具体的なパフォーマンス，可用性，信頼性要件を満たすために，独自のハードウェア装置やプラットフォームによって実現されてきた．その結果，特定ベンダ技術に束縛された，柔軟性を欠くアーキテクチャを継続して利用しなければならない場合が多く，本分野におけるシステムのオープン化，ならびに，期待される急速な技術革新を滞らせてきた一因である．昨今のハードウェアやアプリケーション・ソフトウェアの技術革新と比較すると，本分野のシステム基盤となるソフトウェアプラットフォームは，事実上の標準（de-facto standard）が存在しないのが現状である．OSDL（Open Source Development Labs）が推進する CGL（Carrier Grade Linux）のように，本分野におけるオープン技術は，ようやく具体的なソリューションが世に送り出され始めたばかりである．

第二の課題は，サービス競争が激化の一途をたどる電気通信事業者において，如何にして付加価値の高いサービスを安く・早く提供することで競争に打ち勝つことができるかという点であろう．

とりわけ，IP 化の進展により多様化を極める昨今の通信サービスにおいて，電気通信事業者は他社に先駆けて新しい通信サービスを市場に投入することで，初期需要を刈り取り有利にビジネスを展開できるような市場構造を作り出さなければならない．ネットワークそのものの提供による収益は，より付加価値の高いサービスへと移行しつつあり，同時にパーソナライズされたサービスをタイムリーに提供することが求められる．このように，通信サービス分野でもプロダクトの少量多品種化が進みライフサイクルが短くなってきている今日，サービス開発に伴うシステム化は短期間で確実に実行できなければならない状況になってきている．ここ数年に筆者らが手掛けた OSS 開発を振り返ると，いわゆる「3 ヶ月開発」を現実のプロジェクトとして数多く経験してきており，従来に比較すると明白に短期化が進行している．品質を確保しながら，短納期を達成するためには，プロジェクト運営における工夫やノウハウ活用だけではなく，新しい開発方法論やツールの導入が不可欠であると考える．

なお，TeleManagement FORUM の NGOSS（Next Generation OSS）や OSS/J（OSS Through Java）に見られるように，完成度の高い標準的な参照アーキテクチャが広く公開されているが，依然としてコンセプトモデルが多く実際のシステム開発現場でそのまま利用することができるソリューションは限られている．電気通信事業者の IS 部門やシステムインテグレー

タは，これらの参照アーキテクチャを模索しつつ，実際のアプリケーション開発は従来型のスクラッチ開発を行う場合が少なくない．

2. OSS 基盤の課題と To Be Model とは

OSS とはどのようなシステムであるか，その概要を例に沿って解説し，次世代ネットワークを支援するためにその要件がどのように変化してきているかを考察する．

例えば，携帯電話を新規に購入するときを想定してみる．先ずは携帯電話の小売店に行き，端末を選び，そして，住所・氏名などの顧客情報と共に，利用したいサービスや料金プランを選択し，新規にサービス加入することを申込むであろう．このサービス申込はサービスオーダー（SO : Service Order）と呼ばれており，小売店やコールセンタに設置された専用端末から，先の申込情報をシステムに入力することで，OSS は実際にサービスが利用できるようにネットワークと関連するシステムに対してサービスの設定を行う．図 1 に示すように，OSS の中で，SO 受付からサービス開通までの機能を提供するシステムを広義にはプロビジョニング・システムと呼ぶ．このような，一般消費者向けの携帯電話サービスなどでは，新規サービス加入申込から，数分から数十分で通話などの基本サービスが利用できるようになっている．

プロビジョニング・システムは，SO を受け付けた後，顧客がそのサービスの提供を受けられるかどうかの適正を判断し，ネットワークが利用するデータベースシステムに顧客情報を登録する．続いて，電子メールやショートメッセージなどのオプションサービスをメールサーバや関連するネットワーク機器に対して設定し，料金プランと顧客情報を課金システムに登録する．プロビジョニング・システムは，このような一連のデータ設定をネットワーク機器や関連するシステムに対して行っている．本稿では，サービス開通や変更を行うためのネットワーク機器の設定操作を以降，サービスアクティベーションと呼ぶ．プロビジョニング・システムが処理するデータ量は，例えば全国に展開されたある大規模通信サービスでは，1 時間当り約 3,000 から 4,000 に及ぶ SO を処理しなければならない．また，1 件の SO を処理するためにネットワークや関連システムに対して実行するサービス設定コマンドは，サービス種別に大きく依存するが，少なくとも 20 コマンドから，多いものでは 200 コマンドといった単位にまでなる．

一方，サービス開通後は，顧客のサービス利用量に応じてサービス利用料金を徴収しなければならない．OSS は，図 1 に示すように，顧客のサービス利用量を交換機やネットワーク機器から CDR（Call Detail Record）やパケット情報として採取し，顧客ごとの通話明細情報を抽出する．この機能を広義にはメディエーション・システムと呼び，抽出した通話明細情報は後続の料金計算システムや課金システムに引渡し，月々の料金請求と収納業務に連携されるのである．サービス利用料金の請求・収納のサイクルは通常一ヶ月単位であるが，メディエーション・システムは日々発生するサービス利用量をリアルタイム，もしくは，最長でも日次単位で採取・抽出し通話明細ファイルを作成しなければならない．メディエーション・システムが処理するデータ量はサービスの規模に比例するが, 小規模サービスの場合でも 1 日当り約 2,000 万 CDR，国内最大級の通信サービスでは約 9,000 万 CDR から 2 億 CDR といった単位にまでなる．リアルタイム処理の要件が高くなるほどシステムのハードウェア設備は莫大な規模になり得る．

また，サービスの大動脈であるネットワークにおいては，交換機やルータ，スイッチといっ

たネットワーク機器に故障が発生し，サービス提供に影響がある場合は，即座にそれを検知し対応しなければならない．故障検知，ならびに，その対応は一刻を争うものであり，影響程度と範囲に応じて料金を返還・減額する措置が必要になる場合もある．

OSS は，主にこのような業務システムを提供する電気通信事業者の基幹システムであるが，ネットワークに密接した専門的な技術分野において，具体的なパフォーマンス，可用性，信頼性要件を満たすために，これまで独自のハードウェア装置やプラットフォームを用いて実現されてきた．OSS が取り扱うデータ容量と処理パフォーマンスが桁違いとなる次世代ネットワークサービスを支援するためには，オープンな標準プラットフォームでパフォーマンスを確保しつつ，属人性や専門性を排除した一般的なシステムエンジニアリング技法によって短期間でシステム化することができるシステムアーキテクチャが求められる．

2.1　リアルタイム性への対応

多様化するサービスにおいて，個々のサービスレベルを向上させるために，リアルタイム性への対応が要件として取り上げられ始めており，システム全体で共通に利用できる高速な OSS 基盤が必要になっている．

例えば，プロビジョニング・システムでは，サービス申込みから開通までの時間短縮，ならびに，利用料金をいつでも照会できる機能の提供などは，サービスレベルを向上させる上で重要であり，顧客のロイヤリティを維持するために不可欠である．また，サービス開通時間を SLA（Service Level Agreement）のひとつとして品質基準にしているサービスもあり，処理の遅延は顧客とサービス提供者の両方に不利益をもたらす．

従来の固定電話や専用線サービスでは，一般的にバッチ処理により一括してサービスアクティベーションが行われてきた．しかしながら，昨今では顧客が直接サービス申込や変更，解約といった操作を行えるよう Web インタフェースを公開している事業者やサービスが一般化しつつあり，サービスアクティベーションのターンアラウンドタイムは大幅に短縮されてきている．また，一般顧客からの Web アクセスに応えるために，24 時間 365 日無停止システムであることが求められるケースが多い．

このようなシステムを支援するために，接続したネットワーク装置や外部システムとトランザクションの一貫性を保ちながらリアルタイムで大量オーダーの流通を可能にする高速なアプリケーション実行環境が不可欠である．

一方，ネットワークのフル IP 化に伴い，電気通信事業者が提供する IP 網を利用したサービスは「ベストエフォート型」から「保証型」サービスへ移行してきている．

保証型サービスをサポートする OSS として重要なファクタは，顧客とサービス，および，ネットワークとの関係を適切なデータモデルで保持することである．例えば，ネットワーク上のノード故障が発生した場合は，いち早くその影響範囲のサービスと顧客を紐付けることができなければならない．また，障害状況からサービスが SLA で定めた品質基準を満たしていない場合は料金返還などの措置を正確・迅速に行う必要がある．

複数のレイヤーから構成され，多数のノードによりエンド・トゥー・エンドの接続を実現する IP ネットワーク内で，故障ノードを特定しサービスや顧客への影響範囲を検知するには，大量かつ複雑な管理データのリレーションシップを渡り歩く，パスレングスが非常に長いデータ検索処理が必要になる．一方，顧客からの問い合わせに対して迅速に対応するためには，障

害状況と影響範囲の特定に要する時間の短縮はクリティカルな要件である．このような要件に応えるには，従来のリレーショナルデータベースを利用したアプリケーションレベルのソリューションでは一般には限界があると考えられている．メモリ・アーキテクチャを駆使したインメモリ・データベースやキャッシング技術により，このようなデータアクセス・レーテンシを極小化することが効果的である．

2.2 サービスの多様化 vs TTM 短縮

従来，OSS は電話サービスを中核として電気通信事業者に閉塞したシステムとして発展を遂げてきたが，サービスやビジネスモデルの多様化が進み，ISP，NSP，ならびに，CATV など，異なるサービスプロバイダ間でのシステム連携が増加している．また，サービス主導で構築・統合を繰り返すという特性上，新サービスに対応するたびにネットワーク装置やバックオフィスと連携しなければならないため，システム開発のコスト効率が悪いシステム構造となっている．

昨今，OSS のシステム要素技術は，UNIX をはじめとするオープン系プラットフォームが主流になりつつあり，開発プラットフォームとして Java や C#，.NET などのオブジェクト指向言語が脚光を浴びている．一方，よりシステム連携を容易にする技術として XML や Web サービスを利用してシステム間を疎結合する方式が一般化しつつある．このように，発展と衰退を繰り返す多数の要素技術から適切な選択を行うことは困難であり，現在は適切な選択であっても，数ヶ月先，数年先にはどのように変遷し何が普及するかは想像がつかない．開発者は依然として技術革新への追随に多くの時間と労力を費やしている状況下，業務アプリケーションモデルは，要素技術や特定技術に依存しない共通の表記法によって表現できることが望まれる．また，これらのモデルは，下位の技術が変更された場合も将来にわたって互換性が維持されることが理想である．

OMG（Open Management Group）が参照アーキテクチャとして推進する MDA（Model Driven Architecture）は，このような要件を満たす開発方法論であり，組込みシステムの開発プラットフォームとして具体的なソリューションが数多く輩出されている．OSS の分野に MDA を取り入れることで，他事業者との接続，および，ネットワーク装置やバックオフィスとの連携において，下位の技術を意識することなく業務プロセスを中心にアプリケーションを設計・構築することが可能になる．また，一度構築したアプリケーションモデルは，将来新しい技術を採用する場合も互換性が保たれるため再利用することができる．結果として，システム開発におけるコスト効率を改善することが図れる．

ビジネスモデルの変化に伴いシステム間連携の多様化が進む OSS では，疎結合を促進し業務プロセス主導でアプリケーションを設計・構築・運用するための手段として EAI（Enterprise Application Integration）製品が広く利用されている．EAI 製品を利用することによって，業務プロセスを単一のタスク単位に分割し，ワークフローを使って個々のタスクの実行制御や管理が容易になる．また，メッセージベースのアーキテクチャにより，外部システムやネットワークを渡り歩くような，複雑なインテグレーションを容易に実装することが可能になる．従来の EAI 製品の多くは，1994 年頃より発展を続けている COTS（Commercial Off-The-Shelf）製品の相互連携に焦点を当てた実装であり，近年になってようやくネットワーク機器との接続アダプタを提供しはじめた．しかしながら，ネットワークとの接続が不可欠な OSS

において，ネットワークスピードでシステム連携するためのパフォーマンス要件に応えることが困難なこと，ならびに，ネットワーク機器との接続アダプタが不足していることから，EAI製品だけでは昨今のシステム要件を満たすことは難しい．

また，いわゆる既製品であるCOTSとの連携は，これらのソフトウェア製品のバージョンアップに常に対応した接続アダプタを提供し続けなければならないことを意味する．結果としてアダプタのメンテナンス性を悪化させ，ソフトウェア製品がバージョンアップされてから対応するアダプタの提供まで通常3〜6ヶ月の期間を要する．

このような課題に対応するためには，ワークフローに基づいたプロセス制御・管理機構と，ネットワークに親和性が高く，かつ，短期間に必要なインタフェースのみを安価に提供することができる経済的なアダプタ構築技術を具備したシステム連携環境が理想的である．なお，このような経済的なアダプタの考え方は昨今欧米の電気通信事業者で注目されはじめており，本稿では「要件主導型アダプタ」と呼ぶこととする．

また，特定のネットワーク技術やベンダに依存する傾向が強いパッケージソフトウェア（COTS）は，初期の決まりきった機能を実装する場合はTTM（Time-To-Market）短縮に威力を発揮するが総じて高価である．また，初期導入後のサービス変化に対してハードコード化された機能を改修しなければならないため柔軟・迅速な対応が困難である．

とりわけ，画面インタフェースなど具体的なビジネス要件を実装した上位レイヤーのOSSになるほどこのような傾向が強いと考えられる．プロダクトライフサイクルの短期化が進行する状況下，MDAのように将来の変化に強く，かつ，サービスニュートラルなシステム開発アプローチが利用できることが望ましい．

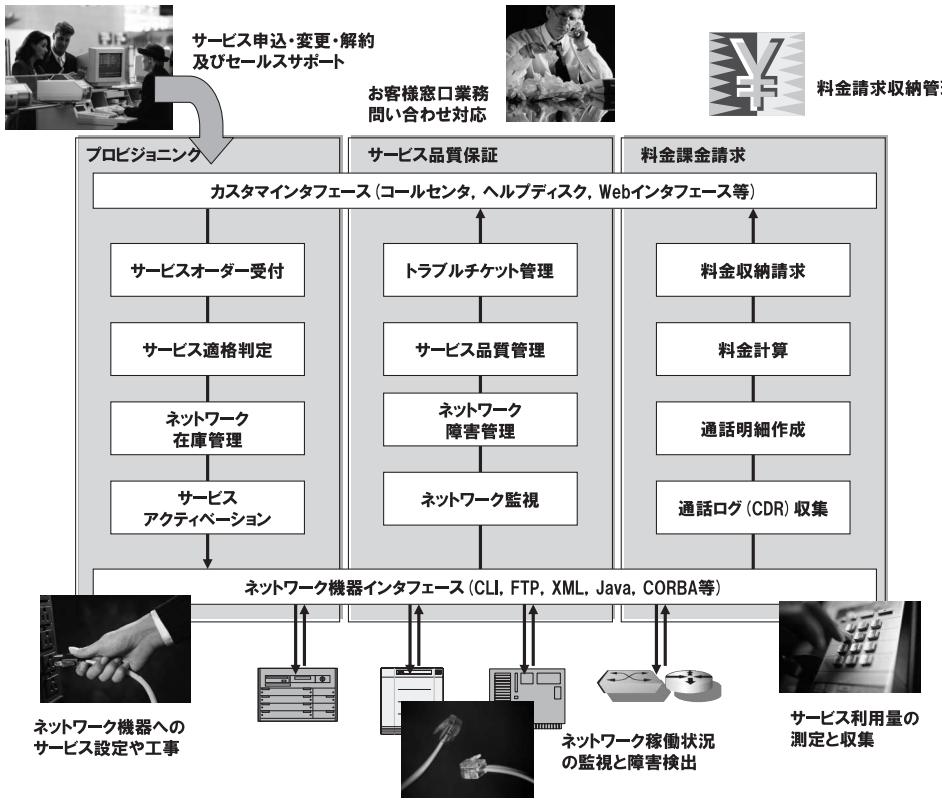

図1　OSSの主要オペレーション構成

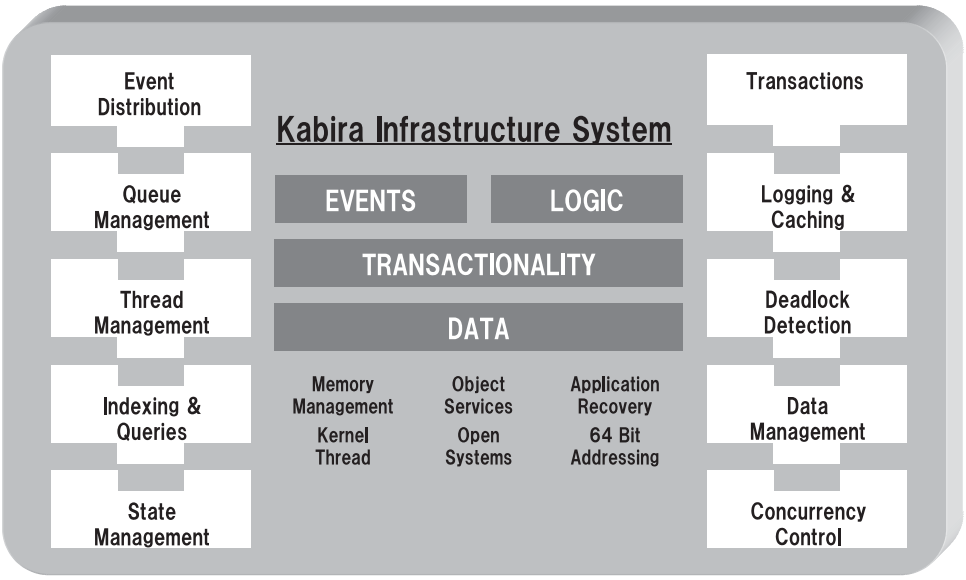

図 2　Kabira Infrastructure Server アーキテクチャ概念

3.　次世代 OSS 基盤製品の実際

これまでに述べた次世代 OSS 基盤の要件に応えるべく，米国 Kabira Technologies, Inc. 社（以下，Kabira 社）は 3 年半に及ぶ研究開発の結果，Kabira Infrastructure Server 製品ファミリ（以下，Kabira サーバ）を世に送り出した．

本章では，Kabira サーバのアーキテクチャ概要を紹介する[2]．

3.1　キャリアグレードの実行環境

Kabira サーバは，アプリケーションの実行プラットフォームであり，すべてのアプリケーション処理はトランザクションのコンテクストが保証される．図 2 に示すようなトランザクション管理機能を提供し，すべてのオブジェクトをメモリ上で処理するアーキテクチャを基本とする．Kabira サーバは，次の各主要機構によってトランザクションの保全性を実現し，高速で信頼性の高いアプリケーション実行環境を実現している．

1）　メモリ常駐型データプロセッシング

コンピュータハードウェア装置において，データを保持し編集処理を行う媒体として，帯域，ならびに，レーテンシ共にもっとも優れているのはメモリ装置である．各ハードウェアベンダ製品の諸元を見ると，メモリ装置/ディスク装置は急速に大容量化が進んでおり，オペレーティングシステムの 64 ビット対応に伴い，アプリケーションから利用できる容量は飛躍的に増加している．

Kabira サーバにおいて，ユーザ・データ，トランザクションステート，ならびにイベントメッセージは，アプリケーションの実行単位であるアプリケーションドメイン*1 で共通の共有メモリ領域に配置される．アプリケーションドメイン内のすべてのアプリケーションプロセスは，この共有メモリ内のデータに対するポインタを保持し，データを直接参照することが可能であり，プロセス間のデータの引渡しに際して IPC(Inter Process Communication)[3]によるデータ移送が発生しない．

このような内部アーキテクチャを採用することで，プロセス間で高速かつ大容量のデータの引渡しを行うメディエーションやプロビジョニングなどの典型的なOSSアプリケーションの実行パフォーマンスを大幅に向上させることが可能であり，実行環境におけるレーテンシを最小化することに一役買っている．

2) オブジェクトキャッシング

Kabiraアプリケーションでは，共有メモリ領域をアプリケーションから見たときの一次キャッシュとして利用することができる．例えば，外部データベースシステムが保存するデータをアプリケーションからアクセスするとき，最初のデータ読み込みはデータベースシステムに直接アクセスするが，2回目以降，同一のデータアクセス要求は，メモリにキャッシングされたデータを参照することで解決する．

キャッシング対象データオブジェクトの決定，キャッシュの更新タイミングは，プログラムのデプロイ時に設定することを可能にしており，設計やハードコーディングとして表現する必要はないため，アプリケーションから透過的に取り扱うことができる．

この機構により，外部データベースシステムからのデータ読み込みパフォーマンスを容易に，かつ，効果的に向上させることが可能になる．とりわけ，顧客マスターデータのように参照を主とするデータアクセスの場合，アプリケーションモデルによってはそのスループットを100倍近い単位で短縮することが可能になる．

また，OSSの典型的アプリケーションのひとつである，CDRメディエーション処理では，途中呼や終了呼の待ち合わせ，ならびに，重複呼を検出するために，このようにメモリ上にキャッシングしたデータを応用することがパフォーマンスの観点から適切なアーキテクチャである．これらの処理では，入力された呼情報をメモリ上の呼と突合させ比較する処理が必要になるが，ディスク装置やデータベースシステムへのアクセスを排除できるため，実行パフォーマンスが飛躍的に向上する．

現在，Kabiraサーバは64ビットオペレーティングシステムをサポートしており，アプリケーションから利用できるメモリ容量が大幅に拡張されている．顧客データのように非常に大規模なデータベースデータを単一のメモリ空間にキャッシングすることも可能になり，更なるパフォーマンスの向上が期待できる．

3) コーディネータサービス

トランザクション処理モニタ[4]として動作するランタイムプロセスであり，Kabiraサーバ上で動作するアプリケーションのトランザクションのコンテキストを保証する．コーディネータサービスは，論理的なアプリケーションドメインをサーバ上に構成し，管理下のアプリケーションプログラムが共用する共有メモリ領域を割り当てる．

コーディネータサービスが管理する共有メモリ領域は，サーバのローカルファイルにマッピングされた「メモリマップファイル」であり，memmapシステムコールを使って割り当てられる．したがって，システムが正常に動作している限りは，メモリイメージはファイルシステム上で永続的に保持されるため，システムやプログラムの停止・再起動によって消失することはない．

コーディネータサービスは，共有メモリ領域にトランザクションログを保持し，トランザクションの一貫性を保証する．何らかの障害によりアプリケーションプロセスが停止した場合，コーディネータサービスはこれを検知し，ビフォアルックスを用いてトランザク

ションをロールバックさせる．次にアプリケーションを再起動し，トランザクションを再開始させる．このようなトランザクションの自動回復機能を具備しており，サーバ単体におけるアプリケーションの動作安定性を高めることができる．なお，コーディネータサービスが障害により停止した場合も，アプリケーションプロセスは独立して動作し続けることが可能である．

4） カーネルレベルスレッドの利用

Kabira ランタイムは，スレッドスケジューリングを行うために POSIX[5]のオペレーティングシステム・レベルのスレッドを使用する．Kabira ランタイムは，各スレッドがそれぞれ割り当てられた処理を終了し，スリープする前に，次に割り当てられるイベントがあるかどうかを検出する．イベントがある場合は，それをスレッドに割り当てることで，スレッドのコンテクストスイッチが発生する機会を最小化するように最適化する．この機構により，オペレーティングシステムの都合等のため不本意に発生するスレッドのコンテクストスイッチを抑制することに成功した．結果として，特定の業務アプリケーション処理に対して，特定のプロセッサを連続して占有使用させることが可能になった．単一のタスクに対して，より多くの CPU 時間を割り当てることができるため，アプリケーション処理に所要する時間を物理的に短縮することができる．

また，アプリケーションプログラムに対する入力イベント数に応じて，割り当てるスレッド数を自動的に最適化する機構を提供する．プログラム初期化時にスレッドプールを確保しておき，必要に応じてウェークアップする方式を採り入れ，プール数を超過するリクエストが入力された場合は，一定の増分プールを追加確保するよう動作する．メディエーションやプロビジョニングのアプリケーションが処理するデータ量やイベント数は，そのときどきの通信サービスの利用量やサービスオーダー数によって変動する．このような特性をもつアプリケーションを動作させるとき，システム資源は自動的に最適化することができる．

なお，Kabira ランタイムは，アプリケーションがデッドロック状態に陥ったときにも，それを自動的に回復することを可能にしている．アプリケーションプロセスはデッドロックを検出すると，そのスレッドを終了させ，デッドロック状態を解放した後に再開始させる．MDA ベースの Kabira アプリケーション開発環境において自動生成したコードはマルチスレッドプログラムであり，開発者は特別な配慮や設計をすることなく単純にランタイム環境にデプロイするだけで，SMP[5]マシンの優位性を享受することができる．マルチスレッドプログラムによって単純に高速化を図るだけでなく，完全に排除することは難しいデッドロックへの対応をランタイム環境が補完している．

このように，実際にアプリケーションが動作するとき，入力データ容量や速度の状況に合わせてシステムがセルフスタビライズする機構を実現しており，一般に高度な技術を要するパフォーマンスチューニングやシステム資源の最適化を容易にしている．この結果，開発者はパフォーマンスやシステム資源の割り当てについて特別な考慮をすることなく，アプリケーションロジックの開発に集中することが可能になる．また，システムのパフォーマンスは，アプリケーションロジックや生成されたプログラムコードに手を入れることなく，単純に CPU 装置を増設するだけで向上させることができる．

Kabira ランタイムはキャリアグレードのプラットフォームとして求められる可用性と

信頼性をサポートするために考え得るサーバテクノロジの集大成であり，メモリ常駐型データプロセッシングと組み合わせることで，高速で安全，かつ，メンテナンス性の高いランライム環境を提供することを可能にした．

5) 分散オブジェクトサービス

ヘテロジニアスな分散環境での相互運用性を可能にする分散オブジェクトサービスを提供する．開発者はサーバの分散やプラットフォームの違いを意識することなく，オブジェクトの分散配置が可能であり，分散したリソースの同期更新やトランザクション境界を透過的に取り扱うことを可能にする．

Kabira 分散環境では，2 つの方式により，分散したオブジェクトの同期更新を行うことができる．ひとつは，ブロードキャスト方式であり，ドメイン内で唯一のマスターオブジェクトへの更新が各スレーブオブジェクトへ連鎖して伝播される方式である．そして，もうひとつの方式はパッシブ方式であり，リモートノード上のスレーブオブジェクトへの更新がオリジナルのマスターオブジェクトへ伝播される方式である．いずれの場合も，伝播ネットワークには，内部的に IIOP プロトコルが用いられており，開発者は，これらの分散オブジェクトは常に最新の状態に更新されているものと見なしてアプリケーションプログラムを開発することが可能になる．

6) HA フレームワーク

分散オブジェクトサービスを応用して，開発者に透過的にオブジェクトのフェイルオーバ・フェイルバックを実現した機能セットである．

このフレームワークサービスでは，複数ノード間でデータのトランザクション一貫性を保証することを透過的にサポートする．あるローカルノードでデータオブジェクトを更新した場合，その更新はひとつ，もしくは，複数のリモートノードで保持する共有オブジェクトに即座に反映される．例えば，ローカルノードが障害により停止した場合，アプリケーション処理はリモートノードへフェイルオーバーし，リモートノードは保持するデータオブジェクトを使って引き続きトランザクションを継続して実行することができる．

昨今のプロビジョニングシステムは，インターネットにより 24 時間いつでもサービス申込みやサービス照会を可能にしているシステムがあり，サービスは無停止であることを求められているケースが多い．一方，従来型のクラスタリングソフトウェアを使った典型的なサーバ冗長構成では，障害回復処理においてディスク装置の再マウントが必要になる．また，構成によっては，待機系サーバにおいてアプリケーションの起動が必要になるため，システムが復旧し運転を再開するまでに数分から数十分間の時間を所要する．HA フレームワークを利用した場合，ディスク装置の再マウントやアプリケーションの起動は不要であり，ダウンタイムを数秒単位に短縮することができる．エンドユーザレベルでは，システム停止をまったく検知することなくサービスを利用し続けることが可能になる．

3.2 MDA を具現化したアプリケーション開発環境

プロジェクトの成功は IDE をはじめとする開発環境の整備に左右されることが少なくない．Kabira は，エンタープライズクラスのアプリケーションを構築するために，MDA に準拠したシンプルで強力な CASE ツールを提供する．

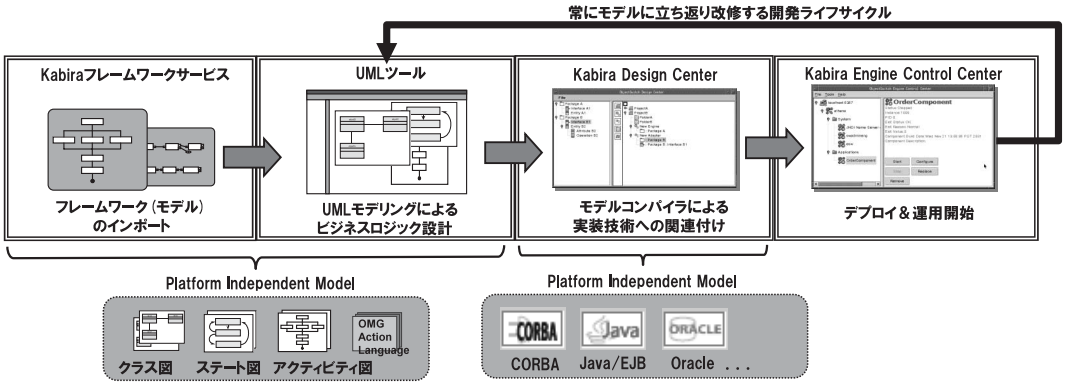

図 3　Kabira 開発プロセスの流れ（設計～製造までの CASE ツールが対応する工程）

1）　標準 UML への準拠

統一モデリング言語（Unified Modeling Language，以下「UML」）を用いて表記された論理的なアプリケーションモデルは MDA の構成要素として不可欠である．Kabira は，標準 UML 表記法によってアプリケーションの設計と実装を支援する CASE ツールを製品化して提供する．

この CASE ツールは，図 3 に示すように，クラス図，ステート図，ならびに，アクティビティ図[*2] を用いて設計したアプリケーションモデルからソースコードを自動生成するモデルコンパイラ機能を提供することで，MDA に準拠した開発プロセスを実現した．

Kabira では，UML の振る舞い仕様記述に Action Semantics を採用した．Action Semantics は，Executable UML を実現するための仕様であり，プラットフォームに依存しない形式でアルゴリズムを記述することができる抽象度の高い記述言語である．この言語を使うことで，開発者は IF-THEN-ELSE といった条件分岐や，WHILE，FOR ループ，データの代入や四則演算を，C＋＋や Java プログラミング言語に近いシンタックスで簡潔に記述することができる．このようにして作成したアプリケーションモデルは，MDA において PIM（Platform Independent Model）[6] に相当する完全に論理的なモデルであり，開発者は下位の実装を意識する必要はない．

2）　モデルコンパイラ

UML と Action Semantics で記述した論理的なアプリケーションモデルを物理的な実装である PSM（Platform Specific Model）[6] に変換する機構を提供する．開発者は，Visual Design Center と呼ばれる GUI ツールを起動し，論理的なアプリケーションモデルに対して，実装技術や要素技術をマッピングさせる作業を行う．通常，この GUI 上にアイコンとして実装されたコンポーネントを使って，アプリケーションモデル内の個々のオブジェクトに対して実装技術や要素技術を割り当てることで物理的なアプリケーションモデルが決定する．

一方，製品が提供するモデルコンパイラは，物理アプリケーションモデルを解釈しソースコードに変換する機能を提供する．モデルコンパイラは，物理アプリケーションモデルから C＋＋コードを自動的に生成し，ターゲットプラットフォーム上でネイティブな C＋＋コンパイラを起動する．その後，コンパイル済みモジュールを Kabira ランタイムモジュールとリンクさせ，実行可能モジュールを生成する．Kabira は，自動生成するプロ

グラミング言語として，その実行速度の優位性からコンパイラ型の C＋＋を選択した．

3)　アダプタファクトリ

外部システムとの接続に必要なインタフェースを手早く構築する「要件主導型アダプタ」をサポートすべく，アダプタを開発する際に最小単位のコンポーネントとなるプロトコルレベルのアダプタを提供する．プロトコルアダプタは，そのインタフェースが UML から直接呼び出すことのできる形式として実装されており，アプリケーションモデルと統一されたルック＆フィールでアダプタが公開する API を利用することができる．

この方式により，対向システムやネットワークとの接続において，必要なインタフェースのみを迅速かつ正確に開発することを可能にした．

4)　フレームワークサービス

Kabira は，OSS に典型的な業務アプリケーションのアーキテクチャモデル，ならびに，共通機能コンポーネントを汎化したフレームワークサービスを提供する．これらのフレームワークを利用することで，実際の開発現場の要件に即したアプリケーションを効率よく構築することを支援する．

【メディエーションフレームワーク】

コンポーネント単位のビルディングブロック工法をサポートする開発フレームワークを提供する．開発者はひとつ，もしくは，複数の機能を MDA ベースの開発環境で実装し，コンポーネント化する．図 4 に示すように，個々のコンポーネントを実行順序に沿って論理的に接続し，CDR 入力から通話明細編集処理，課金システムへの出力に至る一連のメディエーションプロセスを組み立てる．

また，現在最新のフレームワークサービス（KTS：Kabira Transaction Switch）では，アダプタ開発において Action　Semantics による振る舞い記述さえも完全に排除した．ア

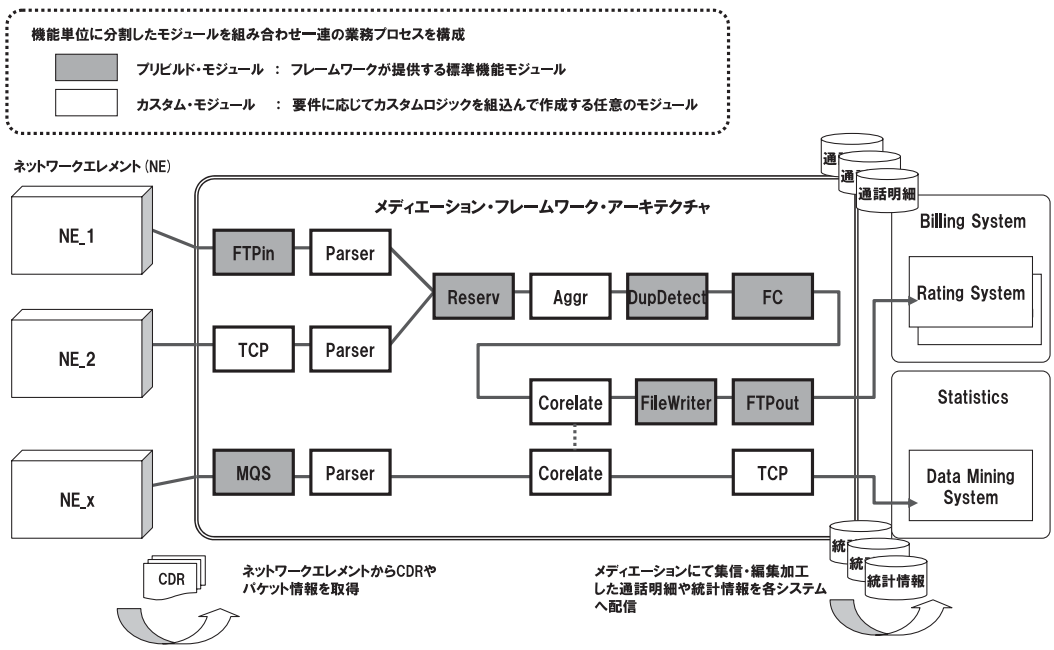

図 4　メディエーション・アーキテクチャモデル概念

ダプタの実装仕様となるインタフェース定義や接続仕様を外部パラメタとして指定することで，実行可能なアダプタコンポーネントを自動生成する機構を開発し提供している．

【プロビジョニングフレームワーク】

ワークフローに基づいて業務プロセスのフロースルー制御・管理を実現するフレームワークである．UML アクティビティ図によってビジブルにワークフローを定義する機構をサポートする．併せて，インタプリタ方式のワークフロー実行エンジンを提供し，定義されたワークフローはコンパイルすることなくそのまま実行可能なプロセスフローとして利用できる．

なお，フレームワークは，ワークフローからネットワーク装置へ接続するために専用のアダプタを提供している．ネットワーク装置にコマンドを送信してサービス設定を行う場合，そのコマンドは一連の複数のコマンドを組み合わせた論理的なコマンドセットである場合が多い．また，接続ネットワークやネットワーク装置そのものの運転状態によって，コマンドが異常終了する場合や無応答となる場合がある．サービス設定のトランザクションの一貫性を保つためには，失敗したコマンドを再実行するか，もしくは，既に実行して正常終了したコマンドをすべて取り消し，サービス設定全体をロールバックする必要がある．専用アダプタでは，図 5 に示すように，このような再実行処理やロールバック処理を UML アクティビティ図やステート図を用いてビジブルに定義し，即座に実行することができる．

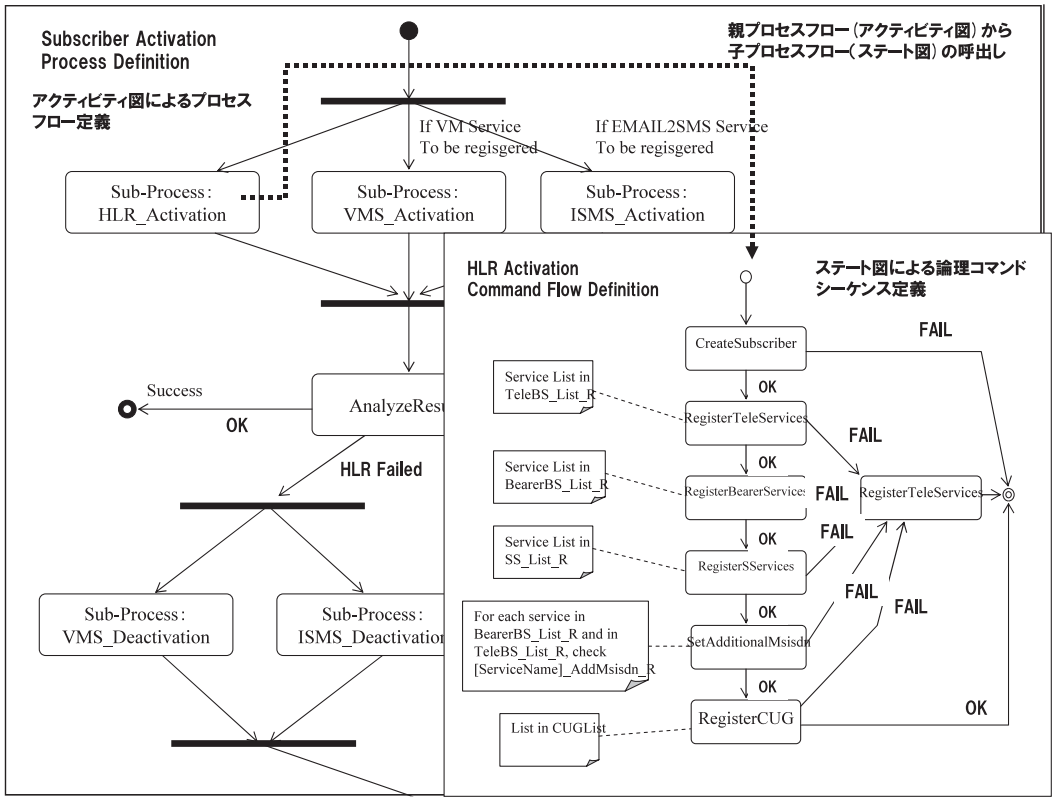

図 5　UML アクティビティ図，ステート図によるワークフロー定義例

4. 次世代 OSS 基盤の導入効果への考察

日本ユニシスでは，電気通信事業者数社において Kabira ソリューションを利用した OSS 開発を手掛けてきた．典型的なメディエーションとプロビジョニングの各業務アプリケーションについて，その導入効果を考察する．

4.1 IP 電話メディエーション

IP 電話サービスを提供する CA 装置（Call Agent）から CDR（Call Detailed Record）を集積し，通話毎に使用明細情報を作成し，後続の料金計算システムに渡す機能を提供する．図 6 に示すような主要コンポーネントから構成されるシステムを，Kabira メディエーションフレームワークを利用して構築した．

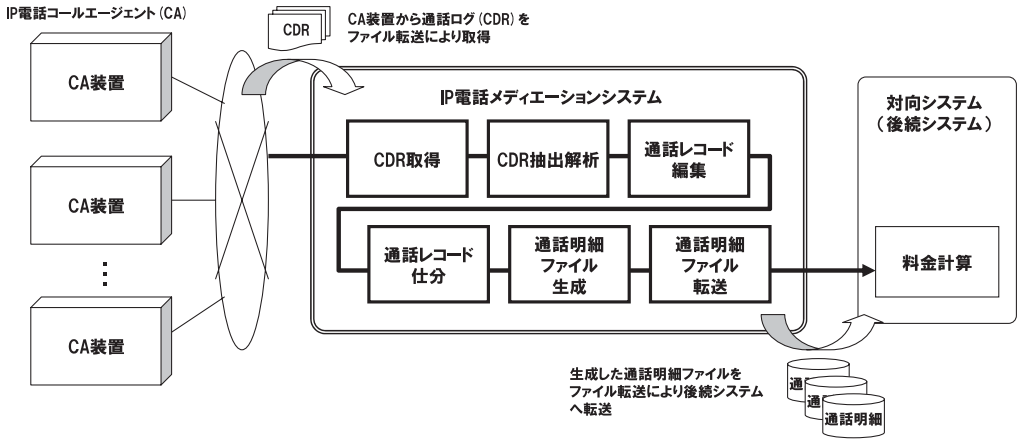

図 6　IP 電話メディエーションシステム・主要コンポーネント構成

1）　短期開発要件への対応

フレームワークサービスが提供する標準機能を最大限利用し，新たに開発する機能を最小化することにより，短期開発への対応を図った．フレームワークサービスは，FTP 転送や CDR 突合処理・重複検査処理など，メディエーション業務を構築するために必要となる典型的なデータ処理モジュールの雛形を与える．これらのモジュールをビルディングブロック工法で組み立てることで，CDR 入力から加工・編集，後続システムへの通話明細ファイル出力までの一連の業務プロセスを容易に構成することができる．

日本ユニシスでは，本事例以前にいくつかのメディエーションシステムの開発において Kabira フレームワークを利用した実績があり，典型的な業務プロセスの設計と実装を資産として有している．メディエーションシステムは，顧客要件によって入力する CDR フォーマットの違いや実行するビジネスロジックの違いこそあれ，基本的なストリームライン処理には大差がないという特徴をもつ．本事例では，このように日本ユニシスに蓄積された資産を可能な限り再利用することで，新たに個別に造り込まなければならない機能を更に絞り込み，結果として短期間・低コストのシステム開発を成功させることができた．

なお，Kabira が提供する Action Semantics は，抽象度の高いロジック記述言語である．

ファンクションポイント当りの生産性は，使用するアダプタに依存するが，Java や C＋＋等のオブジェクト指向言語に比較しても約 1.3〜1.8 倍程度の向上を期待できることが実証された．

2） 保守性の向上

CA 装置のバージョンアップや機能改善に伴い，初期構築以降，出力する CDR フォーマットが数回に渡って変更された．このため，アプリケーションを繰り返し改修しなければならない状況が続いた．

メディエーションフレームワークは，すべての処理が機能単位でモジュール化されているため，CDR フォーマット変更に伴い作り直さなければならない範囲を特定することが容易である．また，モジュール化により，変更による他の既存モジュールへの影響を排除することが可能である．

なお，アプリケーションロジックは誰にでもわかり易い抽象度の高い Action Semantics で記述されているため，初期構築後にプロジェクトメンバーや開発者が変更になった場合も，円滑に技術情報を引き継ぐことが可能になった．

3） 性能要件への対応

従来のメディエーションシステムは，ネットワークベンダや特定の技術者の職人技によって構築されてきたケースが多く，具体的なパフォーマンス要件は，プログラムコードレベルでの緻密なパフォーマンス・チューニングや専門的なノウハウにより対応してきた．MDA ベースでシステムを設計・開発する場合も，通例は自動生成したコードに後から手を入れることでパフォーマンス・チューニングを行うため，結果として，MDA 本来の高生産性を享受していないことが散見される．MDA に基づいた設計・開発アプローチは，Kabira サーバのような高パフォーマンス・プラットフォームと組み合わせることで，初めて真価を発揮できるものであると考える．このことにより，従来は専門エンジニアだけが対応可能であった分野も，一般の IT 技術者により対応可能になる．

Kabira サーバをプラットフォームとして採用することで，メディエーションシステムを構成する各機能モジュールやプロセス間でのデータの移送が発生しない．これらのデータはシステム上で単一のメモリ領域にオブジェクトインスタンスとして配置され，各モジュールやプロセスはメモリ上のデータへのポインタを保持するだけである．このアーキテクチャは，CDR 入力から通話明細出力までの一連のストリームライン処理をもっとも効率よく行うことを可能にするものと考えられる．

このように，Kabira メディエーションフレームワークを利用することで，実行パフォーマンスと短期開発の相反する 2 つの要件を両立できることが実証された．しかしながら，システム構築に当っては次のことに留意しなければならない．

1） パフォーマンスボトルネックの排除

CDR 入力から通話明細出力までの一連のストリームライン処理の中で，パフォーマンスボトルネックとなる箇所を作らないように注意を払うことが重要である．ボトルネックの主たる原因は，大容量データの全文検索やデータの突合など一般にパスレングスの長い処理である．これらの処理は，検索対象のデータをインデックス化する，突合のために比較対象とするキー項目を最小限に絞るなど，ボトルネックを発生させないための工夫が必要である．

また，データをメモリに常駐させることでパフォーマンスを高めることを基本アーキテクチャとするため，ディスクI/Oを伴うオペレーションを極力少なくするように設計することが肝心である．

なお，本事例では，米国Kabira社のエキスパートエンジニアと厳密なコードレビュを実施することで，このようなパフォーマンス・ボトルネックを徹底的に排除することを図った．結果として，当初の性能要件以上の実行パフォーマンスを引き出すことに成功した．

2) ベンチマーク値を使ったサイジング

従来の市販メディエーションパッケージ製品は，どちらかといえばメディエーションのストリームライン処理を構成する論理モジュールの設計や組み立てを容易にする，GUIベースのCASEツールの能力が評価されるものが少なくなかった．このようなアプローチは，開発の短期間化を支援することができるが，実際にできあがったアプリケーションのパフォーマンスが出ないといったケースが多く，パフォーマンス・チューニングに苦労することがある．

一方，Kabiraメディエーションフレームワークを使った開発においても，複雑な論理モジュールをむやみに組み合わせた場合，パフォーマンスが極端に低下する場合がある．組み合わせる論理モジュールの処理形態や処理特性によって，実行パフォーマンスはそれぞれ異なる．また，ストリームライン処理は，パフォーマンスが悪く処理が重い論理モジュールを組み込むと全体のパフォーマンスがそれに引きずられて低下する特性をもつ．論理モジュールの処理形態やCDR項目数ごとに変化するパフォーマンスを事前に測定し数値化しておき，設計時にシミュレーションすることで正確なサーバサイジングを図った．

4.2 IP電話プロビジョニング

IP電話のサービス申し込みをWebサービス，もしくは，オペレータの画面入力によって受け付け，後続のCA装置や関連サブシステムに対してサービスアクティベーションを行う機能を提供する．24時間365日，無停止でIP電話のサービス申込み受付とプロビジョニングを行うシステムを構築した．図7に示すような主要コンポーネントから構成されるシステムを，Kabiraプロビジョニングフレームワークを利用して構築した．

1) 多様なインタフェースへの対応

外部連携システムが多数あり，インタフェースの厳密性が求められたが，フレームワークサービスがバンドルして提供する専用アダプタと論理的なコマンドシーケンスをUMLで表現する技法を利用することで，短期間で確実にインタフェースを構築することができた．外部システムとの連携は，メッセージやコマンドを渡すだけの単純なインタフェースではなく，コマンドの実行結果を待ち受け判定し，その結果に応じて次のプロセスやタスクの実行を制御し，エラー時はコマンドのリトライやエラーメッセージ出力を制御するなど，論理的なプロセスによって実現される．Kabiraプロビジョニングフレームワークを採用することで，UMLアクティビティ図やステート図を用いて，このような論理コマンドシーケンスをビジブルかつコーディングレスで実現することが可能になった．

2) 開発生産性の向上

オーダー流通の優先制御や流量制御，CA装置設定コマンド実行結果の解析とエラー時の自動ロールバック，システム障害時の仕掛中オーダーの回復処理など，フレームワーク

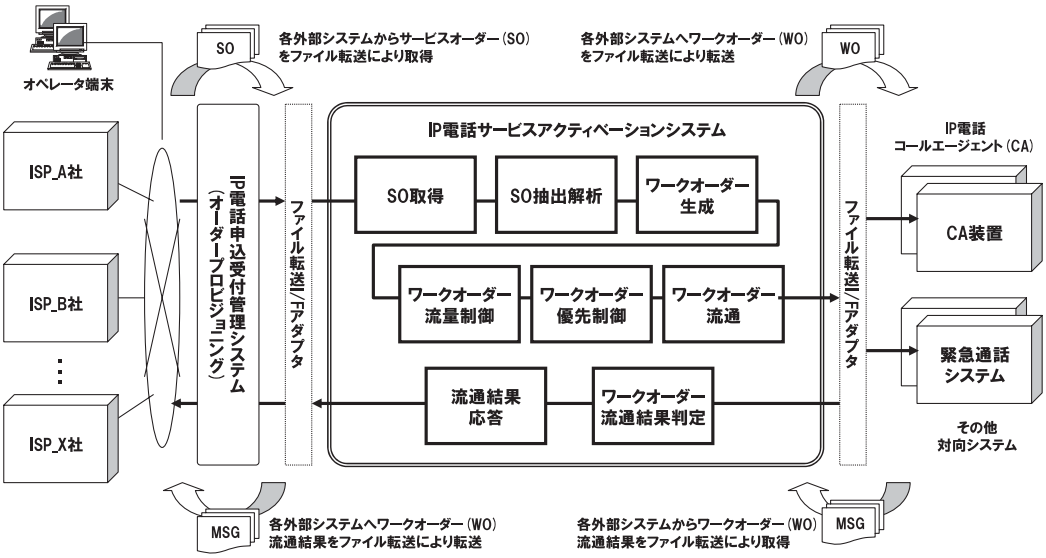

図 7　IP 電話プロビジョニングシステム・主要コンポーネント構成

サービスが提供する標準機能モジュールを最大限利用することで，大幅な工期短縮を実現した．

とりわけ，オーダーの優先制御や流量制御は技術的に難易度の高い実装を伴うため，開発工数を増大させる一因となる．フレームワークサービスが提供する標準機能は，プロビジョニングの業務プロセスフローに直接組み込むことができるモジュールであり，とくに加工することなくそのまま利用できる．

また，プロビジョニングシステムでは，システム障害が発生した場合，その瞬間で走行中のオーダープロセスは，個々のオーダーによってそれぞれ異なる状態になる．フレームワークサービスは，仕掛かり中オーダーのスナップショットを周期的にダンプする機能を標準で提供している．システム障害によって，サーバ装置が待機系に切り替わった場合，もしくは，サーバ装置が障害から復旧した場合は，このスナップショットをシステムへリロードすることで，障害発生直前のトランザクション状態にシステムをロールバックすることが可能である．このようなシステムのリカバリー設計と実装は一般に高価であり，技術的な難易度も高い．製品の標準機能によって，障害発生時にそれぞれ異なる状態を保持するオーダーを完全に回復することが可能であり，このようなリカバリー処理を行うための設計や実装は不要である．

IP 電話サービスは，提供する付加サービスによってプロバイダとして他社との差別化を図る必要がある．CA 装置の機能拡張に伴い，プロビジョニングシステムでは新規サービスに対応したオーダー情報の管理や CA 装置へのコマンド設定追加等の制御が必要になる．本事例では，プロビジョニングシステムの初期構築後にキャッチフォン等の付加サービスの追加対応が必要になった．日本ユニシスは，付加サービスの追加対応に短期間で対応できるように，オーダー情報の管理機構をプロビジョニング・プロセスフローから独立した外部ファイルによって行えるように工夫した．このことにより，新規の付加サービスへの対応が必要となった場合も，プロビジョニングシステムは約 2〜3 週間程度の短期間で対応できるようになった．

なお，要件定義による業務分析の結果，プロビジョニング・プロセスフローが産出されるが，このフローは UML アクティビティ図を用いてストレート，かつ，ビジブルに業務プロセス実装に反映することができる．UML で表記した業務プロセスはそのまま実行可能なワークフロー定義であり，コーディング作業を排除することで工期短縮を図れるだけでなく，運用後のサービス変更や新しいプロセスやタスクの追加といった，保守性も大幅に向上させる．

例えば，CA 装置へサービス設定コマンドを送信し，その結果を受信し判定する一連の論理的コマンドシーケンスは，次のようなタスクを UML アクティビティ図で表現することで，実行可能なワークフローを組み立てることができる（図 8）．

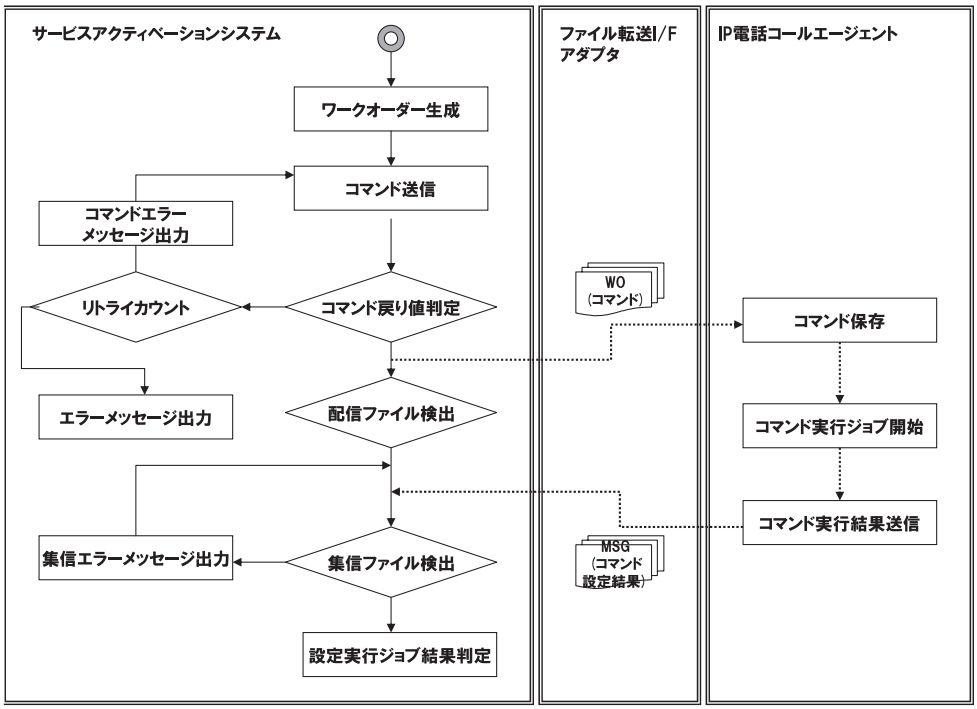

図 8　サービス設定コマンド送受信処理フロー定義イメージ

このように本事例では，Kabira プロビジョニングフレームワークの能力を最大限利用することで通常のスクラッチ開発の標準的な工期の約半分の期間でシステムを構築することに成功した．ただし，実際にプロビジョニング・プロセスフローを構築する際は，次の点に留意すべきである．

1）　接続ネットワーク仕様への厳密な対応

ネットワーク機器や外部システムと接続するために構築したインタフェースは，実際にネットワークを介した接続試験を実施しなければその動作の正当性を確認することはできない．インタフェースは，このような外部システムや機器とのインタフェース仕様書に基づいて設計し構築するが，例えば，機器が古くて仕様書が陳腐化している，あるいは，既存のネットワーク環境に依存する特別な設定が必要である，など，仕様書だけでは情報が不足している場合が多々ある．結果として，構築したインタフェースを使って実機を接続

したときに，想定外の動作をしたり，不具合を検出したりすることがある．

Kabira プロビジョニングフレームワークを利用して構築したシステムでは，このような場合も UML で表現したシーケンスやパラメタ設定記述を手直しするだけで実際の接続仕様に合わせてインタフェースを改修することが可能になるため，手戻りの発生を最小限に食い止めることが図れる．しかしながら，プロジェクトを推進するときには，このような想定外のエラーに対応するためのスケジュールを予め確保しておくことが望ましい．また，想定外エラーに対する開発スケジュール遅延やコストオーバといったプロジェクト・リスクを最小にするために，対向システムやネットワーク機器のシミュレータを並行開発し，これを用いて充分に結合試験を実施することが有効な手段のひとつである．

4.3 課題と処方

これらの事例をとおして，本稿の冒頭に述べた 2 つの課題に対して Kabira のソリューションが有効な処方として適用できることが実証された．また，Kabira の専用フレームワーク製品は，それぞれの業務アプリケーションを構築する上で最適なアーキテクチャ設計と機能モジュールを与える Best Practice であることが確認できた．

もっとも，システム構築に際しては，このようなツールやソリューション製品の機能や能力だけでなく，これらを使いこなしシステム開発プロジェクトを円滑に推進させるソフト面でのサポートが必須である．

1） パフォーマンス要件への確実な対応

パフォーマンス要件に対して確実に対応してゆくためには，システムの屋台骨となるプラットフォームレベルからパフォーマンスに優れたソリューションを適用することが賢明である．しかしながら，プラットフォームはあくまでもシステムの礎であり，その上に業務要件を単純に上乗せしただけでは，システムとして稼動したときに本来のパフォーマンスを発揮することはできない．

良い素材と環境を活かして完成度の高いシステムを構築するためには，これを使いこなすことの出来る総合的なインテグレーション技術と OSS ドメインにおけるシステム構築ノウハウの応用が不可欠である．

2） 短期開発への対応

MDA を用いた設計開発アプローチとそれを支える CASE ツールにより短期開発のハード面は充分補えることが実証された．しかしながら，これを実際のミッションクリティカル・アプリケーションの開発現場で使いこなすためには，技術者のスキルや熟練だけではなく，プラットフォームやフレームワーク製品の開発ベンダとの技術交流，実践的なプロジェクト管理手法の導入など，ソフト面からの支援を欠かすことはできない．

これらの事例では，日本ユニシス標準のプロジェクト管理手法を用いて，徹底した納期管理，変更管理，および，リスク管理を実践することで，プロジェクトとして成功に導くことができたといえよう．とりわけ，短期開発プロジェクトにおいては，数日間の手戻りが納期を遅らせる，プロジェクトのコストオーバを招くといった致命的な損失をもたらす場合がある．すべての想定し得るリスクに対して可能な限り予測可能性を高めることが必要である．

5. おわりに

本稿では，ユビキタスネット社会の実現に向けて急速に発展しつつある次世代ネットワークの舞台裏で，通信サービスの屋台骨となる OSS に求められる要件の変化を考察した．次世代ネットワークサービスを提供するために，OSS は，システムの実行速度，ならびに，開発速度の両面において，リアルタイム性が求められ始めていることは明白であり，ソフトウェアプラットフォームとして，これらの要件に総合的に応えてゆくソリューションが必要になってきている．また，これまでマスサービスを中心としてきた通信サービスは，より付加価値が高いパーソナライズされたサービスをより短いサイクルで提供しつづけなければならない状況へと移行しつつある．

具体的なソリューションとして紹介した米国 Kabira 社のサーバ製品は，このような「高速・大量のリアルタイム・イベント処理」と「短期間化が進むシステム開発サイクル」といった，互いに相反する要件を両立させることができる OSS 基盤ソフトウェアである．OSS 分野のみならず，同種の課題を抱えるクレジットカード与信業務の分野においても活用されはじめており，米国 VISA カードの与信電文中継ゲートウェイとしてサービスを開始したことは記憶に新しい．米国 VISA カードでは，2007 年までに 40,000 トランザクション/秒の処理を計画しており，この要件に応えることができるソリューションとして Kabira を選択した．

本稿で述べたように，リアルタイム性への対応は，そのプラットフォーム製品の能力に依存するところが大きいといえよう．また，昨今増え続けている短期開発への対応も選択する IDE 製品の機能と能力にその成功の鍵があるといっても過言ではない．しかしながら，これらの要件を両立した完成度の高いシステムを構築するためには，このようなツールやソリューションを活用してプロジェクトを効率よく運営してゆくための知恵と工夫が必要である．優れた IDE や開発方法論の登場は，いたずらに短期開発を冗長してはいないか．CASE ツールなどハード面の発展だけでなく，短期開発に備えたプロジェクト管理方法論やソフトウェア工学などソフト面のパラダイムシフトについては今後も検討の余地があると考える．

＊ 1　アプリケーションドメイン：システムを構成するアプリケーションプログラムの集合のことであり，Kabira サーバのデプロイする実行環境の単位．
コーディネータサービスは，アプリケーションドメインごとに共有メモリ領域を確保し，アプリケーションのデータやオブジェクトを配置する．

＊ 2　アクティビティ図：アクティビティ図は，Kabira プロビジョニングフレームワークが提供するワークフロー機能にて利用する．

参考文献
［1］ 総務省，「情報通信白書平成 17 年版」，第 1 章　特集「u-Japan の胎動」
［2］ Dan Sifter，「The Kabira Infrastructure System Architectural Overview」，2002
［3］ Maurice J. Bach，「UNIX カーネルの設計」，共立出版株式会社，1993 年 11 月 15 日
［4］ 日経 BP 出版センター，「オープン OLTP システム入門」，1995 年 12 月 7 日
［5］ ビル・ルイス＋ダニエル・バーグ，「P スレッドプログラミング」，株式会社ピアソン・エデュケーション，1999 年 1 月 10 日
［6］ 和田　洋，安竹　由紀夫，「MDA（Model Driven Architecture）と現実の開発プロセス」，ユニシス技報，通巻 81 号，2004 年 5 月，P.49

執筆者紹介　井　上　　純（Jun Inoue）

1964 年生．1989 年松山大学経営学部卒業．同年日本ユニシス・ソフトウェア(株)入社．金融システム向け外接通信ミドルウェア，データベース管理システムの設計開発業務を担当後，オープン OLTP 製品や各種ミドルウェア製品のテクニカル・マーケティングに従事．現在，日本ユニシス・ソリューション(株)テレコムビジネス部にてプロジェクトマネージメントに従事．