今後のシステム開発技術の展望

The Past and The Future of System Development Engineering

羽 田 昭 裕

要 約 ソフトウェアの特質は複雑性と柔軟性であると言われる．プロダクトとプロセスにおける過去の開発技術は，これらの特質に対応して発展してきた．その技術の中で，1）アプリケーション設計とソフトウェア・アーキテクチャを分離するアーキテクチャの基本型，2）アーキテクチャ中心の開発プロセス・フレームワークが主要な技術である．

現在のシステム開発のフレームは，コンピュータのネットワーク化と分散オブジェクト技術であり，その鍵となるコンセプトはサービス志向である．このコンセプトを実現するには，現在の開発技術にはさまざまな技術的課題がある．

システム開発方法 LUCINA のコンセプトを中心に，本特集号の論文を紹介し，システム開発技術の将来を展望する．

Abstract Complexity and Flexibility are main features which make production of software difficult. Architecture-centric software process framework and reference architecture which separate the software architecture from the application design are key technologies to achieving these required features.

Current frame of system development is networked computer and distributed object technologies. The key concept of the frame is service—oriented. The concept causes various challenges to the key technologies.

With describing concept of LUCINA method and topics of this technology review, this paper tries to make proposal to system development in the future.

1. はじめに

コンピュータの発達と利用の広がりに開発技術が対応できることを目指して，ソフトウェア工学は生まれ，発展してきた．現在の課題は，グローバルにネットワーク化したコンピュータと，産業や日常生活に入りこんだ利用方法に対応したシステム開発技術を提供することであろう．本特集は，このような課題に対応した経験と知見の一部を集成している．

システム開発とは，あるシステムを対象として開発作業を行なうことである．開発する対象であるソフトウェア・プロダクト，開発過程であるソフトウェア・プロセスのいずれにおいてもアプリケーション・フレームワーク，プロセス・フレームワークなどフレームがキーワードとなっている．本論も「フレーム」を切り口にシステム開発技術を見ていく．フレームは，ある特徴を共有する問題を一般化したものである．そのフレームに合致するものが数多くあれば，そこで使われた手法と技術の適用が期待できる，というのが要点である[24]．

システム開発技術の一つとして，日本ユニシスでは開発方法 LUCINA[*1] を提供している．本稿では，LUCINA のコンセプトを中心に，システム開発技術の展望を述べる．まず，システム開発のフレームとしてのコンピュータ・ネットワークと分散オ

ブジェクト技術の歴史を振り返り（2 章），そのフレームが開発技術に与えた影響を概観する（3 章）．続いて，LUCINA を中心に現状のシステム開発技術に対する基本的な考えを述べ（4 章），その基となる具体的な知見を示す各論文を紹介する（5 章）．

2. システム開発のフレーム

ネットワーク化されたコンピュータに求められることは，欲しいサービスを欲しい人に，適切なタイミングで提供することである．これはネットワーク・コンピューティングとして提唱されてきたことである．ここでは，コンピュータ・ネットワークと分散オブジェクト技術が，フレームとしてどのように準備されてきたかを振り返る．

2.1 コンピュータ・ネットワーク

コンピュータ・ネットワークが発展した背景として，次のように消費者，提供者の変化があると言われている．

1） 消費者の変化

ネットワーク化されたコンピュータを対象とする情報通信業界は，より多くの人が使うことで利用価値が高まるというネットワーク外部性[*2] をもつとされる．多くのコンピュータがネットワーク化されることにより，データやプログラムといったソフトウェアが交換できるようになる．インターネットによりコンピュータ間の接続が標準化され，ネットワークに接続される環境が発展した．ネットワークの影響は，ネットワークの大きさの 2 乗の勢いで広がる（「メトカルフの法則」）といわれるように，自己増殖的に普及するインターネットにより，この傾向は加速された．同時に，一般の利用者の割合が増え，ユーザビリティが重視されるようになった．

2） 提供者の変化

利益構造もネットワーク型になってきた．一般に，製品は普及し，質がよくなり製品寿命が長くなると，需要が供給を下回るようになる．代表的な製品の利益構造は，同一の製品を大量に作る「規模の経済」から，いくつかの消費者セグメントに対応した一連の製品群を提供する「範囲の経済」へと進んだ．しかし，このような対応は製品の飽和を加速したため，次にサービス化，個人単位でのカスタマイズ，技術革新による製品ライフサイクルの短縮化が目指された．ライフサイクルの短縮化を実現するために，従来の垂直統合型の企業モデルから，企業間のネットワークによって対応する「連結の経済」を意識した水平統合型へ変化した．水平統合型では標準，特に企業間のネットワークを実現する標準が重要になる．情報通信業界では，この傾向が顕著である．

このようにインターネット技術を背景として，コンピュータがネットワーク化すると共に，製品ライフサイクルの短縮化，サービス化が進行した．これに対応し，システム開発の分野では，オープン・アーキテクチャ化した[*3]．

2.2 分散オブジェクト化

以上のような経緯をコンピュータ・システム利用者の観点から見ると，サービスとは，ネットワーク化されたシステムの利用者が主体となるアプリケーションとなる．ネットワーク化されたシステムとは，アプリケーションがどこに存在しているか，ど

のように実装されているか（言語，ミドルウェアなど）を意識すること無く利用できるという透過性（transparency）を実現したものである．このような透過性は，CORBA[*4]，J 2 EE[*5]，COM＋[*6]などの分散オブジェクト技術で実現されている．また，利用者が主体となるシステムでは，アプリケーションの所有者は利用方法を制御できず，利用者が利用のシナリオを組み立てる主体となる．そのためには，アプリケーションの側が個々の利用者と文脈を認識し，利用者に合わせた振る舞いをする必要がある．

このようなサービスを実現するには，アプリケーションをモジュール化し，独立性を持たせるために，オブジェクトとして実現することが必要となる[18]．そしてハードウェア，ネットワークだけでなくソフトウェア側でも，オープン・アーキテクチャが必要となる．

これに対応し，80 年代末のオープン・アーキテクチャの基本型（reference model）に続き，分散オブジェクトモデルの基本型も，OMA（Object Management Architecture）[27]（図 2），RM-ODP（Reference Model for Open Distributed Process）[23]という形で示された．このように，オープン化，ネットワーク化が進展し，それをオブジェクト指向技術が支えるという形で進行している．

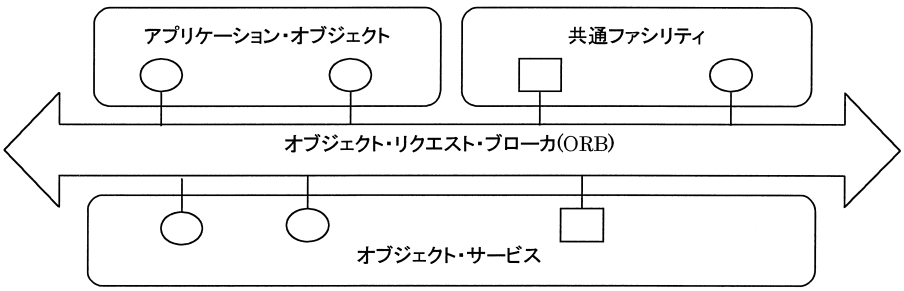

図 1　OMA（Object Management Architecture）[27]

例えば，OMA の基本型は，ORB をソフトウェア通信バスとして，アプリケーション固有のアプリケーション・オブジェクト，複数のアプリケーションで利用するサービスである共通ファシリティ，これら全ての分散オブジェクトが利用するオブジェクト・サービスで構成している．

3.　開発技術の課題―フレームに欠けているもの―

日本ユニシスでは，CORBA を分散オブジェクトの基盤として大規模分散システムの開発を手がけてきた．その経験から，分散オブジェクト環境での開発技術の補強が必要であるという認識に至った．ここでは，開発技術を開発対象（ソフトウェア・プロダクト），開発過程（ソフトウェア・プロセス）という 2 面から見る．

3. 1 ソフトウェア・プロダクト

ソフトウェア・プロダクトを，アプリケーションとアーキテクチャに分離することは，分散オブジェクト技術の特徴である[13]．ここでは，アーキテクチャの面，特に層

別のアーキテクチャと大規模分散に対応した設計方針に注目する．具体的には以下の通りである．

3.1.1 ソフトウェア・アーキテクチャとは

ソフトウェア・アーキテクチャとアプリケーション設計の分離する方法に基づく開発ではソフトウェア・アーキテクチャを再利用し，それに当てはめてアプリケーションを設計する[*7]．これは，要求からアーキテクチャとアプリケーションを毎回一から設計していた，従来の開発方法と異なるものである．

例えばデータベース設計の分野では，リレーショナル・データモデルなど特定のフレームを前提としたモデルを論理モデル，前提としない概念モデル，特定のプロダクトを前提とした物理モデルに分離していた．これに比べ従来のシステム開発方法は，論理モデルを，実装環境を特定しない「理想的な実行環境」でのモデルとして，フレームを示さなかった．ソフトウェア・アーキテクチャは，この「理想的な実行環境」を明示するものと考えられる．

ソフトウェア・アーキテクチャ論では，CORBA と同様のアプローチが主流となっている．SEI[*8] のプロダクト・ライン・アプローチ[10]では，アーキテクチャを共有する同一系列のアプリケーション群（プロダクト・ライン）を，再利用の対象として重視している．

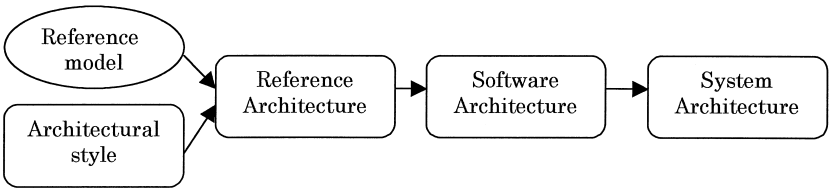

図 2　各アーキテクチャの関連[24]

ここでソフトウェア・アーキテクチャに関する用語を整理する．現在のところ標準化した定義はないが，SEI[14]は次のように示している（図 2）．基本型は構成要素（software component）の機能を表す．アーキテクチャ・スタイル（architectural style）は，バッチやクライアント・サーバといったスタイルであり，構成要素と構成要素間の制御とデータの流れのパターンを示す．基本形にアーキテクチャ・スタイルという制約を加えたのが，基本アーキテクチャ（reference　architecture）である．これを実際のシステムの構造として示したものがソフトウェア・アーキテクチャである．ソフトウェア・アーキテクチャおよび，それに実行環境・開発環境を加えたシステム・アーキテクチャをアーキテクチャと呼んでいる．

3.1.2 層別のアーキテクチャ

ところで，アーキテクチャはシステム制御構造・アプリケーション制御構造・アプリケーション構造に層別できる[4]．これは OMA では，オブジェクト・サービス，水平共通ファシリティ，垂直共通ファシリティにほぼ対応する（図 3）．現在，オブジェクト・サービスの一部と ORB のみが実現され，他は実用化されていない．言い換えれば現状では分散オブジェクト技術で提供されるアーキテクチャは，システム制御

構造を示し，その上下の階層へのインタフェースのみ提供している．この意味で，アプリケーション設計を分離できるアーキテクチャを示せていない．これは，ドメイン・アプローチ，プロダクト・ライン・アプローチでも同様であり，標準的な位置に立つドメイン・モデルを示していない．元来，プロダクト・ラインは（供給側の）企業内の再利用であり，企業を超えた再利用と区別している．

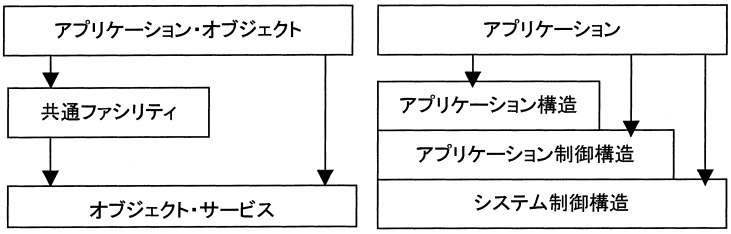

図 3　OMA と層別アーキテクチャ

また，分散オブジェクト技術の利用が進むにつれ，いくつかの欠点が明らかになった．主なものは，巨大な仮想メモリ上に同格のオブジェクトがメッセージ交換しているというメタファにリアリティが無いことである．現状からすれば，アプリケーション内部，アプリケーション間，企業内，企業間でのアーキテクチャ的な区別が必要である．また，実行環境は実現されたが，運用環境に関する方針がないなどの点もクローズアップされた．

こういった状況から，分散オブジェクト技術の適用方針として，アプリケーション間では標準的な分散オブジェクト技術，企業間ではインターネット技術を用い，アプリケーション内・企業内では企業毎にプロファイルを持つ方向に進んでいる．プロファイルとは，OMG などの標準化団体で活用されている手法であり，用途などの類型により標準を具体化したものである．例えば，飛行機を製造するプロジェクトと事務作業をシステム化するプロジェクトでは，異なるプロセスやシステム・アーキテクチャを定める．

3.1.3　設　計　方　針

CORBA などの分散オブジェクト技術は，システムの構造を基本型として示しているのみである．この構造に適したシステムを作成するためには，どのように動作させるかというアーキテクチャ・スタイルに加え，どのように作るのか，どのように配置するのか，を考える必要がある．これを提供するのが設計方針（design strategy）である．

これまでの方法論や設計技法は，アプリケーションの内部構造に注力していたため，アプリケーション間の通信を重視している分散オブジェクト技術に対しては不十分である[13]．例えば，オブジェクト指向の開発方法論は，CORBA のオブジェクト・サービスで提供されるデータベース，トランザクション，ネットワーク等を正面から取り上げていない．先に述べた，アーキテクチャとアプリケーションの分離の状況からすれば，プロファイルを作成し，標準とプロファイルを利用した設計方針が必要となる．

3.2 ソフトウェア・プロセス

ソフトウェア・プロセスの面では，ライフサイクル・モデルへの批判が 80 年代初頭から出始めた[30]．批判の基本は，ライフサイクル・モデルは，ハードウェア生産のモデルであり，柔軟なソフトウェアの開発にはそのまま適用できない，ということであった．ソフトウェアの柔軟性とは，要求を初期に纏めることができず，要求が変化するという点である*9.

3.2.1 ソフトウェア・プロセスの確立まで

1） プロトタイピングと操作的な仕様

ソフトウェアの柔軟性への対策として，プロトタイピングが提案され，操作的な仕様がそれに対応した．

例えば,ライフサイクル・モデルへの批判と,操作的な仕様を方法化した JSD[32] では，”与えられた仕様を満たすだけでは不十分であり，利用者の追加要求に応えられなければならない．利用者の追加要求は事前に予測できない．そのため利用者の視点でモデル化することから開発を始めるべき”と主張している．使い捨て（mock-up）のプロトタイプは，振る舞いを模擬して，作りはどうでも良い，ということである．また，骨格型は，いつでもシステムが完成品のミニチュアであることを基本としている[31]．このように，プロトタイプは，使い捨てにせよ，骨格型にせよ，振る舞いを仕様化し，仕様からシステムを生成することが基本となる．

2） オブジェクト指向技術

オブジェクト指向技術は，このような流れに対応して発展した．しかし，プログラミング言語，方法論，実行環境が示されるだけで，それらを利用したプロジェクトをどう計画し，見積もり，進捗を把握し，納品し，運用・保守していくか，ということが欠けていた．文献[19]に従えば，OO 開発方法論は表 1 のような経緯を辿った．

表 1　オブジェクト指向方法論の歩み[19]

	方法論(Methodology)の定義	代表的な方法論
1960 年代末	プログラミングレベル：規約,Tips,ヒント集	
1970〜1980 年代	設計レベル：技法,ガイドライン,支援文書集	CRC
1990 年代初期	特定プロジェクトのライフサイクルを扱う方法集	OMT,OOSE,...
1997〜	プロセス・フレームワーク	OPEN,RUP,...

3） ソフトウェアプロセスモデル

他方，システム開発技術の側では，80 年代末にプロセス・プログラミング[29]というアプローチが始まった．これはソフトウェア開発のアナロジーで，ソフトウェアプロセスの開発・保守を行なうというものである．この考え方をプロセス定義という形で発展させた能力成熟度（CMM； Capability Maturity Model）[26]がプロセス評価の標準となっている．また，B.Boehm の新しいライフサイクル・モデル[28][15]，COCOMO II[16]の見積もり技術がなどが登場した．そして，これら

の技術に加え，建築業でのベスト・プラクティスを参考にしたPMI（Project Management Institute）のプロジェクト管理技術[21]，作業者の創造性を重視するマネージメント技術が導入されてきた．

また，これらの研究から組織のプロセスの重要性が認識された．プロセスを定義することで，各々のプロジェクトで得た経験が組織のプロセス改善に繋がり，品質などを測定する基礎となるからである．一方，プロセスを定義することは手間がかかり，それぞれのプロジェクト毎にプロセスを新設することは実用的ではない[26a]．プロジェクトというのは，定義*10からしても反復されないものであり，プロセスも組織の文化やプロジェクトの性格に左右されるからである．そこで，汎用的に示しながら，カスタマイズ可能な開発プロセス・フレームワークの標準化が進んだ*11.

3.2.2 ソフトウェア・プロセスの動向

このようにソフトウェア・プロセス研究が進む中，アーキテクチャがシステム開発の成否を握ることなどが，ケーススタディから明らかになり，ウォータフォール型とスパイラル型の長所を取り入れたアーキテクチャ中心の反復型開発が提案された[15]．

反復型開発とは，一回の開発を複数のリリースで組み立て，それぞれの新しいリリースを，前回のリリースの骨組みをさらに肉付けしていくかたちで洗練していき，それぞれのリリースにおいて，当面するリスクに応じて，どのような部分を拡張していくべきかを決めていく開発方法である[22]．

この結果，ライフサイクル批判に始まった研究機関を中心に提案された手法のうち，操作的な仕様はシナリオを中心とした振る舞い分析*12，インタフェース中心のコンポーネント・ベース開発となり，プロトタイピングを取り入れた開発は，アーキテクチャ中心の開発*13として実用化された．

このような流れの中で，オブジェクト指向の方法論も，プロセス・フレームワークに変化した*14．RUP（Rational Unified Process）[8]，OPEN（Object-oriented Process, Environment and Notation）[19]などがその例である．

ソフトウェアの柔軟性に対応するこれらの手法の必要性が高まっている．これは2章で述べたような状況から近年のシステム開発は，開発期間・システムの構造・ライフサイクルなどのプロジェクト構造が，従来の経験の単純な延長ではなくなっているため，と考える．

ところで，プロセス・フレームワークは，それぞれのプロジェクトにカスタマイズ可能な形で提供されており，実際のシステム開発へ適用するには抽象的であった．また，新しいプロセスを採用するには，それを実行するためのプロジェクト管理技術が必要である．ソフトウェア・アーキテクチャ論の帰結からすると，具体的でカスタマイズでき管理可能という意味で実用的なプロセスは，基本アーキテクチャあるいはプロファイルに合わせたものになる，と考えられる．

4. LUCINAのコンセプト

このような状況に対し，当社は社内外での成功事例，研究成果を基に開発技法LUCINAとして纏めた．端的に言えば，LUCINAの目的は，問題指向の開発を実現す

ることであり，そのコンセプトの基本は，実装をシンプルにすることとコンポーネント指向である[*15]．

問題指向とは，変化していく要求に注目することである．ソフトウェア開発はリスクへの対応が主となるので，何かへの注目を減らすにはそれがリスクにならないようにする必要がある．問題への注目を実現するためには，解（＝実装）への注目を減らすことが重要であり，これを実現するには解法をシンプルにする必要がある，と考えた．そのための手段が標準への準拠とプロファイル化である．ドキュメント量およびプロセスを構成するアクティビティを減らすことが，シンプルさの指標である．

1)　プロファイル化

LUCINA のプロファイルは，基本アーキテクチャとそれ対応したシステムアーキテクチャで構成される．プロファイル化の目的は，アーキテクチャの再利用である．技術のライフサイクルが短縮する中，実装をシンプルに保つにはアーキテクチャの持続的な改善と再利用が必要である．また，プロファイル化によりアーキテクチャの検証（proof of concept）も容易になると考えている．アーキテクチャの再利用プロセスを，ABC（アーキテクチャ・ビジネスサイクル）[14]に基づいて描くと以下の手順になる（図 4）．

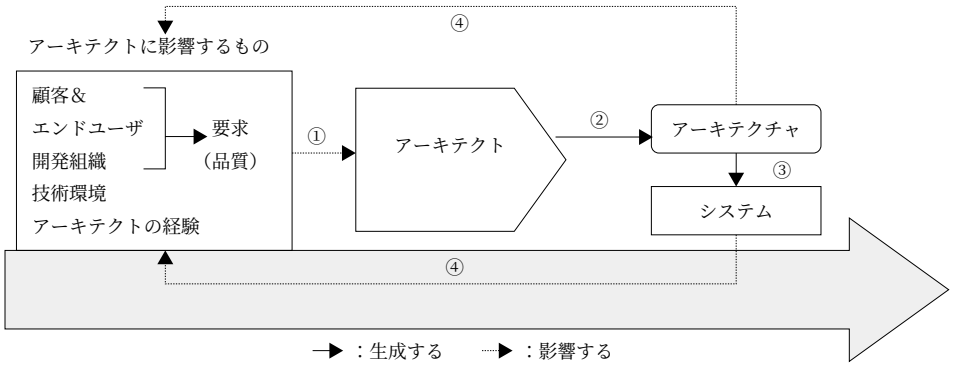

図 4　ABC（Architecture Business Cycle）[24]

①　アーキテクチャを構想する
②　アーキテクチャを分析し生成する
③　アーキテクチャからシステムへ
④　アーキテクチャの再利用

このプロセスの中で，アーキテクチャの構想に役立ち，実際のシステムからフィードバックを受けるのに充分な具体性を持つことをプロファイルの要件としている．

プロファイルに対応させて，設計方針を提示することで，中間成果物であるドキュメントを少なくすることや，アーキテクチャを実装者の共通理解としやすくすることも狙っている[*16]．

2)　標準への準拠

標準への準拠は，オープン・アーキテクチャでの新技術への対応方法として常

道である．

アーキテクチャの標準としては，CORBA，J 2 EE，COM＋などの分散オブジェクト・モデルとインターネット技術に対応している．HTTP ベースのプロトコル利用を主眼として，TCP/IP ベースの CORBA のアーキテクチャを発展させたのアーキテクチャになっており，アプリケーション内，アプリケーション間，企業間に分けた層別のアーキテクチャとそれに対応した設計方針を示している．サービスの利用はシン・クライアントベースとしている．

プロセスの標準としては，共通フレーム[*17]に対応している．また，OMG で策定中のプロセス・モデルの動向も注視している．

3） コンポーネント指向

ところで，要求への集中は反復的な開発を必要とする．反復的な開発は，プロジェクト管理の負荷を増大させる．反復的な開発とプロジェクト管理の適正な負荷を両立させるために，コンポーネントのアイデアを拡張し，プロダクト（実行単位），プロセス（開発単位），ピープル（チーム構成）を対応させている．これがプロセスをスケーラブルにする鍵と考える．

5. 各論文と流れ

ここまで見てきたように，過去 10 年の動向は，デジタル化とネットワーク化に支えられている．これに対応する開発技術は，プロダクトの面では，アーキテクチャの重視，特に分散オブジェクト技術であり，その想定がインターネットにより見直されたこと，プロセスの面では，プロセス・フレームワークやプロジェクト管理技術の発展が特徴であった．

LUCINA はこのような要請に対するシステム開発技術の一部分である．本特集号で取り上げた論文は，LUCINA を支える社内外の成功事例である．内容的には，LUCINA の範囲に関わるものと，LUCINA を補完するものがある．

5.1 LUCINA 関連

5.1.1 ソフトウェア・アーキテクチャ

1） アーキテクチャの構想・作成

ソフトウェア・アーキテクチャで重要なプロセスは，アーキテクチャ構想の過程である．論文「LUCINA for Java/Web Logic Server の構造」では，ユーザ中心のナビゲーション（画面遷移）とユースケースの関係を考察している．Web アプリケーションは，従来の GUI ベースのアプリケーションに比べ，開発が困難である．第一に，利用する技術が多く，しかも新技術が多い点である．第二に，ナビゲーションの主体がユーザとなる，コンテンツと表示方法が分離される，などに対応したデザイン・スタイルが未確立であること[8]が大きい．一方，現在提供されている方法論やプログラミング・モデルは暗に GUI ベースのデザイン・スタイルを仮定している．この論文では，LUCINA としての模索のポイントを示している．この考察から，ユースケースの役割は追跡可能性の保障であり，導出できることと追跡できることは異なることを示している[*18]．

論文「リース業パッケージ開発における HYPERPRODUCE Ⅱ 適用での考察」

は，オフィス・サーバで生かされたパッケージ開発技術を，Windows　NT 環境で活かした事例を示している．オフィス・サーバでのパッケージは，カスタマイズ容易なプロダクト・ラインを従来から実現している事例と考えられる．アプリケーション制御層で隠蔽した開発・実行環境の必要性，そのような環境には製品としての持続性が必要であることを示している．

2)　基本アーキテクチャ

アーキテクチャの構想で参照されるのが基本アーキテクチャである．Java は，Web アプリケーションを構築するための標準的な言語となっている．しかし，J 2 EE では基本型のみが示されていた．論文「J 2 EE ブループリンツにおける多層アプリケーションの設計ガイドライン」は, J 2 EE を基本型にし, MVC（Model-View-Controller）などのアーキテクチャ・スタイルを取り入れて，Sun が具体化した設計ガイドラインを紹介している．

3)　アーキテクト

基本アーキテクチャが示されたとしても，アーキテクチャの構想と作成はアーキテクトの創造的な作業である．

アーキテクトがどのようにアーキテクチャを構想したらいいか，そのアプローチを紹介しているのが論文「オブジェクト指向技術の実践適用」である．この論文は，既存の技術者を念頭に，継承や多態性を重視した初期のオブジェクト指向方法論や，データ中心アプローチと対比し，実際にユーザに適合された事例に基づいて，現代的なオブジェクト指向開発の鳥瞰を示している．

また，論文「新技術者像と教育」はアーキテクトをどのように要請していけばいいかを考察している．この論文では，新入社員をアーキテクト候補へ育成するための教育アプローチに焦点を当てている．

5.1.2　ソフトウェア・プロセス

1)　プロセス・フレームワーク

論文「ラショナル統一プロセス」[1]は，代表的な現代的なプロセスである RUP を紹介している*19．RUP はアーキテクチャ中心のインクリメンタルな開発に適合する，カスタマイズ可能なプロセスを実現している．歴史的経緯と実例に加え，よくある誤解に対する丁寧な解説により，RUP の全体像を示している．

2)　プロセスの自動化

RUP もそうであるが，反復型開発にはプロセスの自動化が必要である．プロセス自動化の対象は，プロジェクト管理，開発支援，開発作業に分けられる．従来は開発作業の自動化が重点であったが，現在は開発支援に重点が移っている．これは，実行環境の抽象度が上がったことに加え，作業者を管理の対象と見るマネージメント観から，作業者の創造性を活かすため，支援をするというマネージメント観への推移も背景にある．論文「チーム開発モデルと TeamFactory」は，チームでの開発を支援する環境である TeamFactory を利用して，作業者が参加するプロジェクト管理プロセスを自動化するための方法を，LUCINA のチーム開発支援モデルと合わせて示している．

3)　クロス開発支援環境

大規模でミッションクリティカルなシステムに反復型開発を行なうには幾つかの課題がある．実運用の実行環境を開発時の実行環境に持ち込むことや，分割した他のサブシステムを開発時に利用することが困難な点である．このようにミッション・クリティカルなターゲット環境と開発プラットフォームが異なるためクロス開発を支援する必要がある．論文「プロトタイプ手法を支える SPIRIT 開発ツール」は，TUXEDO ベースで，これらを実現した開発環境を示している．

5.2 LUCINA を補完するもの

5.2.1 システム・アーキテクチャ

アプリケーションがサービス化する場合，アプリケーションとシステム・アーキテクチャの分離が重要である．特にインターネットを利用したシステムでは，専用線，携帯電話など接続形態が多様な上，サービス要求の量が予測できず，要求量の増大に合わせて随時システム拡張するなど従来と異質な信頼性や高可用性が求められることから，アプリケーションをその基盤から切り離す必要性が高い．

論文「ASP サービス基盤「Kiban@asaban」」は，ASP（Application Service Provider）のシステム基盤である kiban@asaban の設計思想を示している．システム基盤は，システム・アーキテクチャを実現し，実行環境と運用環境を提供する．この基盤上でのアプリケーション設計方針を LUCINA が提供しているという関係になる．サービス指向を実現する先進的な形態である AIP（Application Infrastructure Provider）の現状の課題と方向性も示している．

基盤のうち，最も重要な要素がセキュリティである．論文「情報セキュリティ対策検討のために」は，セキュリティについて要素技術とセキュリティ・ポリシーの両面から解説している．

これらの論文は，基盤提供者とサービス提供者の区分が明確になっていき，基盤においても，その構造や機能が変化していることを示している．

5.2.2 データの情報化

顧客毎にカスタマイズしたサービスを提供するためには，基幹系のデータを情報系に如何に繋げるかが当面する課題である．この問題を，データウェアハウスとデータ・マイニングという二つの側面から検討している．

サービス志向のシステムでは，利用者がサービスを発見すると同様に，提供者側も個々の利用者を認識し，それにあったサービスを提供する必要が生じる．論文「顧客分析とデータ・マイニングの動向」は，統計手法を体系化して個々の顧客を分析するデータ・マイニングを実現する方法を示している．統計解析はモデル記述の形式化，問題と解の分離などにつき，長い実績のある分野である．問題指向でアプローチし，データの情報化，アプリケーションのサービス化，開発プロセスの自動化を実現している例となっている．マイニングのリアル化，非数値データへの適用についての考察を含め，問題指向のアプローチにモデル化が重要な働きをすることを示している．また，これを実現するためには，基幹系，情報系の区別なく，顧客を軸にデータを扱える環境の必要性を示している．

このように，サービス志向のシステムでは，基幹系，情報系の連携が必要となる．このための技術がデータウェアハウスである．基幹系システムは，トランザクション

を中心にするため，効率と整合性が重要になる．リレーショナル・データモデルは，それに適するように設計されている．一方，意思決定のための情報系では，データを多面的に検索する必要がある．このために，マルチ・ディメンジョナル・モデルなどが提供されている．論文「データウェアハウスのモデリング」は，これを実適用するための，課題とアプローチを示している．

6. おわりに

ACM[20]や情報処理学会[7]の学会誌での将来予測も，ネットワークやコンピュータについての技術動向と，それらが人間社会のあらゆる側面に深く入り込むことを述べているが，それに対応するシステム開発技術の展望は示していない．本稿ではフレームをキーワードに，システム開発技術の課題について考察した．M.Jackson は，フレームを使うことで特化した問題に集中することができ，その結果，開発技術は工学化し，要求や問題を中心とした開発になると主張する[25]．D.A.Norman は，ユーザビリティの立場から，人間中心の開発にするためには，コンピュータが特化したアプライアンスに変わること，そして反復的な開発が必要であると主張している[17]．

ところで，技報の 20 年の歩みは，ちょうどソフトウェア・プロセスの発展時期と重なる．この間技報を育んできた柳生孝昭氏や山﨑利治氏の論述を含む技報 60 号の論考は，問題とシステム開発を結ぶ仕様記述とモデル化について追求している．この号の基調をなす古村哲也氏の課題設定に倣えば，今回の特集の課題はネットワークコンピューティングに重点を置き，アプローチはプラグマティックである．この傾向は，コンピュータのネットワーク化と深く関係している．そこに，どのようなフレームが必要かは未知であり，サービスを支えると期待されるエージェント技術なども研究の域にある．

このような状況の中，今後のシステム開発を支える技術者および技術者を育てる方々に，本特集号が少しでも参考になるとすれば，望外の喜びである．

＊ 1　LUCINA は日本ユニシスが提供する開発方法．アーキテクチャ，開発プロセス，開発ツールのセット．

＊ 2　外部性（externality）とは，市場外での経済主体の直接的な関係により，財・サービスの効用（utility）などが増加・減少すること．効用とは，消費者が主観的に感じる満足度合い．ネットワーク外部性とは，消費者が財・サービスから得られる効用が，その財・サービスに加わる消費者の数が増えるに従って，増大すること．

＊ 3　オープン・アーキテクチャ（open system architecture）とは，社会的に共有されたインタフェース仕様に基づき，独自に設計されたコンポーネントや製品を協動させるためのアーキテクチャ[14]．

＊ 4　CORBA（Common Object Request Broker）は，標準化団体 OMG（Object Management Group）が定める分散オブジェクト基盤の標準仕様．URL : < http : //www.omg.org/ >

＊ 5　J 2 EE（Java 2, Enterprise Edition）は，SUN が定める Java によるサーバ・サイド・コンポーネントのフレームワーク．

＊ 6　COM＋（Component Object Model）は Microsoft が定めるコンポーネント・ソフトウェア技術の総称．

＊ 7　これは他のエンジニアリングと同様の考え方である．ソフトウェア以外のエンジニアリング分野で「再利用」という言葉は使われず，標準的なコンポーネントを体系的に設計して利用する[34]．元来，アメリカ的生産方式では，基本システムの共通性の上に立って，部品の互換性と相互運用を基本にしてきた．

＊ 8　SEI（Software Engineering Institute）は，米国防総省の支援によりカーネギーメロン大学が運営するソフトウェア研究組織．URL：〈 http：//www.sei.cmu.edu/〉

＊ 9　ここでいうライフサイクルとは，ウォータフォール型の開発プロセスのことである．このプロセスは，構造化の手法を前提としている．ライフサイクル批判を始めた一人である M.Jackson の構造化に対する批判は次の 2 点に要約できる．①ソフトウェアの柔軟性ゆえ，問題全体を把握してから，それを分解したうえ，ソフトウェアを構成するという構成法が取れない．②複雑さを，単一な構造に分解しようとする構造化ではなく，直交したモジュールのネットワークによって実現する方が柔軟性を保てる[24]．このコンセプトを実現する方法として提出された JSD は，CSP（Communicating Sequential Process）[33]を「理想的な機械」として，仕様を直接実行する方法を示した．この後，M.Jackson は，JSD 自体を相対化し，問題フレームというアプローチを提出した．

＊10　PMI では，プロジェクトとは，“ 独自性のある成果物やサービスを創出するために遂行される有期の活動 ” と定義している[21]．

＊11　例えば，COCOMOII は，2000 年代には異質な 5 つの開発集団となることを想定して，調整可能にしている．5 つの集団とは，エンド・ユーザ，コンポーネント開発者，コンポーネント統合作業者，システムインテグレータ，インフラストラクチャ開発者である．

＊12　“ 振る舞い ” とは，システムの外部から観察することができ，検査ができるシステムの働きである．従来から振る舞いを仕様として表現するアプローチに対し，シナリオで表現することが主流となってきている[9]．ユースケースは，システムの振る舞いをシナリオの集合として分析する場合に，シナリオを体系化するための道具である．

＊13　G.Booch[22]は，プロジェクトが置く焦点を，カレンダー，要求，文書，品質，アーキテクチャに分類し，アーキテクチャ主導の方法が望ましい，と述べている．例えば，要求主導型は，機能というシステムの外部的な特徴に注目するため，保守しにくいシステムを導きやすいとしている．他のアプローチも，同様としている．但し，人命に関わるようなシステムでは，品質主導のアプローチを推奨している．

＊14　記法が UML（Unified Modeling Language），手法がデザイン・パターンとして確立し，方法論の対象としてこの領域が残ったことも背景と考える．

＊15　LUCINA のアーキテクチャ[3]と開発プロセス[2]の概略については，過去の技報を参照頂きたい．

＊16　要求への集中と，ドキュメントの最小化は，近年登場した軽量プロセス（lightweight process）の目標と重なる．XP（eXtreme Programming）[5]や FDD（Feature-Driven Development）[11]がその代表である．これらのプロセスは，アーキテクチャを軽視しているように見えるが，アーキテクチャとアプリケーションとの分離を前提として，アプリケーションの開発技法を示している，と解釈している．

＊17　共通フレームとは，ISO が定める SLCP（Software Life Cycle Processes）を基にした SLCP-JCF 98[35]のこと．

＊18　この問題に対しては，ユーザビリティの観点を開発プロセスに導入する[12]，Web 独自のアーキテクチャに適した UML プロファイルの標準化[6]というアプローチがされている．いずれも体系的な導出の手順を示せていない．また，コンテンツと表示形式を分離するための技術も発展途上であり，解決策は模索中といえる．

＊19　LUCINA は，RUP と同様に Boehm[15]のフェーズに沿った，アーキテクチャ中心の開発プロセスを持ち，それは RUP 5.5 以降のプロセスと類似している．

参考文献

［1］　野田他，“ ラショナル統一プロセスソフトウェアのベストプラクティス ”，技報 68 号，日本ユニシス，2001.

［2］　羽田，“ 青山学院バーチャル・キャンパス基盤システムの構築 ”，技報 65 号，日本ユニシス，2000.

［3］　羽田，“ コンポーネント指向の Java アプリケーション開発技法 ”，技報 63 号，日本ユニシス，1999.

［4］　石田，“ 基幹系向けミドルウェアのオブジェクトモデル ”，技報 63 号，日本ユニシス，1999.

［5］　K. Beck, eXtreme Programming：Embrace Change, Addison-Wesley, 2000.

［6］　J. Conalllen, Building Web Applications with UML, Addison-Wesley, 2000.

［7］　“ 情報処理技術―過去十年そして今後十年― ”，情報処理，Vol. 41, No. 5, 2000.

［8］　Jacob Nielsen, Designing Web Usability, New Riders Pub., 2000.（邦訳：篠原監訳，ウェブ・ユーザビリティ，MdN コーポレーション，2000）.

［9］　J. M. Carroll et al, “ ユーザの視点を取りいれる技術：システム開発におけるシナリオの役割 ”，情報処理，Vol. 41, No. 1, 2000.

［10］　P. Clements, L. M. Northrop, A Framework for Software Product Line Practice-version 2.0, SEI, 1999.

[11] P. Code, et al, Java Modeling in Color with UML : Enterprise Component and Process, Prentice-Hall, 1999.（邦訳：依田訳, Java エンタープライズ・コンポーネント, ピアソン, 2000）.
[12] L. L. Constantine & L A. D. Lockwood, Software for Use, Addison-Wesley, 1999.
[13] T. J. Mowbray, W. A. Ruh, Inside CORBA : Distributed Object Standards and Applications, 1997, Addison-Wesley.（邦訳：大谷ら監訳, Inside CORBA, 1998, アジソン・ウェスレイ）.
[14] L. Bass, et al, Software Architecture in Practice, Addison-Wesley, 1998.
[15] B. Boehm,“ Anchoring the Software Process ”, IEEE Software, Vol. 1 3, No. 4., 1998. URL : 〈http : //sunset.usc.edu/publications/TECHRPTS/1995/usccse 95-507/ASP.pdf〉.
[16] “ COCOMO II Model Definition Manual ”, University of Southern California, 1998. 〈http：//sunset.usc.edu/research/COCOMOII/index.html〉.
[17] D. A. Norman, The Invisible Computer, MIT Press, 1998.
[18] S. Gossain, Object Modeling and Design Strategy, Cambridge Univ. Press, 1998.
[19] I. Graham, et al., The OPEN Process Specification, Addison-Wesley, 1997.
[20] “ The Next 50 Years ”, CACM, Vol. 40, No. 2, 1997.
[21] PMI, PMBOK（A Guide to the Project Management Body of Knowledge）, PMI Standards Committee, 1996.（邦訳：田中弘他訳, プロジェクトマネジメントの基礎知識体系, エンジニアリング振興協会, 1997）.
[22] G. Booch, Object Solutions : Managing the Object-Oriented Project, 1996, Addison-Wesley.（邦訳：石川訳, オブジェクト・ソリューション, アジソン・ウェスレイ, 1998）.
[23] ISO, “ Reference Model for Open Distributed Processing ”, International Standard 10746-1, ITU Recommendation X.901, 1996.
[24] M. Jackson, Software Requirements & Specification, 1995, Addison-Wesley.（邦訳：玉井・酒匂訳, ソフトウェア博物誌―世界と機械の記述―, 1997, トッパン）.
[25] M. Jackson, 田村訳,“ 問題, 方法そして特化 ”, 情報処理, Vol. 36, no. 1, 1995.
[26] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber,“ Capability Maturity Model, Version 1.1, " IEEE Software, Vol. 10, No. 4, July 1993, pp. 18-27. URL : 〈http : //www.iijnet.or.jp/sea/CMM/〉（邦訳：〈http : //www.iijnet.or.jp/sea/CMM/〉）.
[26 a] W. S. Humphrey, Managing the Software Process, Addison-Wesley, 1989.（邦訳：藤野訳, ソフトウェアプロセス成熟度の改善, 日科技連 1991.）
[27] OMG, The Common Object Request Broker : Architecture and Specification, John Wiley & Sons, Inc., 1992.
[28] B. Boehm, “ A Spiral Model of Software Development and Enhancement ”, Computer, Vol. 21, No. 5, IEEE, 1988.
[29] L. Osterewil, “ Software Process Are Software Too ”, Proc. Of 9th International Conf. On Software Engineering, 1987.
[30] 有澤誠, ソフトウェアプロトタイピング, 近代科学社, 1986.
[31] P. Zave, “ The Operational versus Conventional Approach to Software Development ”, CACM 27-2, 104-118, 1984.
[32] M. Jackson, System Development, Prentice Hall, 1983.（邦訳：大野・山崎監訳, システム開発法 JSD, 共立出版, 1989）
[33] C. A. R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.（邦訳：吉田訳, CSP モデルの理論, 丸善, 1992）.
[34] I. Jacobson, Software Reuse-Architecture, Process, Organization for Business Success, ACM Press, 1997.（邦訳：杉本他監訳, ソフトウェア再利用ガイドブック, トッパン, 1999.）
[35] SLCP-JCF 98 委員会, 共通フレーム 98―ソフトウェアを中心としたシステム開発及び取り引きのための共通フレーム―, 1998, 通産資料調査会.

執筆者紹介　羽　田　昭　裕（Akihiro Hada）

1984年一橋大学卒業．同年日本ユニシス(株)入社．意思決定支援ソフトウェアの開発・適用に従事．その後，業務システムとその基盤の要求分析・開発に従事．現在，Eビジネス技術部コンポーネント技術室に所属．情報処理学会会員．