LUCINA for .NET によるアーキテクチャ中心のプロジェクト・アプローチ

An Architecture-Centric Project Approach using LUCINA for .NET

今 道 正 博, 猪 股 健 太 郎

要 約 近年のシステム開発ではアーキテクチャを確立させることの重要性が増している．不適切なアーキテクチャはしばしば想定外の事態を引き起こし，工数超過，納期遅延などの問題へと発展する．アーキテクチャを正しく確立するためには，多様なリスクに柔軟に対応するためのベストプラクティスが必要である．アーキテクチャを確立させるためのベスト・プラクティスのひとつと考えられる手法が，システム開発の早期にアーキテクチャを確立し検証するアーキテクチャ中心アプローチである．しかし，システム開発のたびにアーキテクチャを作り直すことはリスクを伴う．そのリスクを軽減するにはアーキテクチャの再利用が有効であると考える．

本稿では，アーキテクチャ中心アプローチの概要を紹介した後，日本ユニシスの開発方法である LUCINA for .NET が，アーキテクチャの再利用のために提供するアーキテクチャ・テンプレートについて解説し，アーキテクチャ・テンプレートがシステム開発のどの部分を改善するかを述べる．そして最後に，LUCINA for .NET の拡充のための日本ユニシスの取り組みについて紹介する．

Abstract Recently, it becomes more important to establish an appropriate architecture during software development. An inadequate architecture often causes the unexpected situation, and raises problems such as an overrun of cost, schedule, etc. To implement both software and system architecture suitably, so-called "best practices" with which developers can address various risks flexibly are needed. And one of the "best practices" is "Architecture-centric Approach", a method for establishing and testing architecture in an early phase of software development. But, to repeat designing and implementing architecture from scratch in every project entails some kind of risks. So, for reduction of the risks, the architecture should be reused if at all possible.

First, this paper describes an overview of Architecture-centric Approach. Next, it presents a problem for application of this approach to middle-large scale software development. Then it introduces "Architecture Template", a solution for the problem based on LUCINA for .NET, which is a software development method in Nihon Unisys, Ltd. Finally, this paper summarizes Nihon Unisys's activities to upgrade LUCINA for .NET.

1. は じ め に

システム構築の現場ではしばしば想定外の事態が発生し，工数超過，納期遅延などの問題へと発展する．その原因はさまざまであるが，一般的なケースとして以下が挙げられる．

1) 各種プロジェクト管理係数の把握が不十分
2) 顧客との要求整理・確認が不十分
3) アーキテクチャの確立内容とその検証が不十分

4) ハードウェアの性能検証が不十分

5) 実施テスト内容が不十分

本稿では，これらの原因のうち特に 3) について取り上げその対応策を考察する．

まず，本稿ではアーキテクチャという用語を「コンピュータ・システムの品質に影響を与える設計の主要な要素」と定義する．コンピュータ・システムの品質とは，レスポンスやスループットに代表される性能や，スケーラビリティ，可用性，保守性，信頼性，セキュリティを意味する．そして，これらに影響を与える設計の主要な要素とは，コンピュータ・システムを構成する部品の種類，部品間のインタフェース，内部構造，全体を特徴付ける要素技術である．

アーキテクチャを確立する際には，コンピュータ・システムが要求される品質を満たすことを示さなければならない．そのためには，機能や論理構造，それらを実現するサブシステムやコンポーネントの構成，ネットワークトポロジ，物理的な分散や配置の仕組みまでが明確に規定されていなければならない．

システム構築では，めまぐるしい環境の変化，外乱要因，多様なリスクに柔軟に対応しながら，上記のように抽象的な概念であるアーキテクチャを正しく可視化し，充分検証しなければならない．よって，それらを効果的に実現する手法，アーキテクチャ確立の「ベストプラクティス」というべき手法を構築プロジェクトへ効率的に適用する必要がある．

続く 2 章では，必要十分なアーキテクチャを開発プロジェクトで実現するための最適な手段として，ラショナル統一プロセス（RUP）で採用されている『アーキテクチャ中心アプローチ』という開発手法を紹介し，その利点を述べる．3 章では，アーキテクチャ中心アプローチの課題について述べる．4 章では，日本ユニシス（以下，当社）の開発方法である LUCINA for .NET が，アーキテクチャ中心アプローチの課題をどのように補完し，システム開発の生産性向上や品質向上に寄与するかを述べる．そして 5 章では，LUCINA for .NET の拡充のための日本ユニシスの取り組みについて紹介する．

なお，上記のケース 1) 2) および 4) 5) については，各要素に最適な解決策（プロジェクトマネジメントなど）が存在するため，本稿では言及しない．

2. アーキテクチャ中心アプローチとは

本章では，システム開発の早期にアーキテクチャを実装し，検証することを可能にする代表的な手法としてアーキテクチャ中心アプローチを選び，その概要と利点を紹介する．

2.1 アーキテクチャ中心アプローチの概要

アーキテクチャ中心アプローチは，コンピュータ・システムの品質に対する要件や実行基盤から受ける技術的な制約を反映したアーキテクチャの定義とそれを実現する実行可能なサンプルプログラムの両方をシステム開発の早期に作り，それを残りの開発における基本パターンとしてコンピュータ・システム全体の構築を進めていく開発方法のことである．

アーキテクチャ中心アプローチは，繰り返し型の開発プロセスであるラショナル統一プロセス（RUP）で採用されている開発方法である．

2.2 アーキテクチャ中心アプローチを採用した開発

アーキテクチャ中心アプローチでは，アーキテクチャの定義とそれを実現する実行可能なソ

フトウェアの両方をシステム開発の早期に作成する．アーキテクチャ作成は以下の手順で実施する．

① 要件定義内容から技術リスクにつながる重大な要件を抽出する
② リスク回避の可能性が高いアーキテクチャの候補を定義し，アーキテクチャ作成の基本的な方向性を決める
③ 品質に対する要件に関連が深い機能を少数選んで実行可能なサンプルプログラムを作成し，設計が要件を満たすことを検証する（プロトタイピング）
④ プロトタイピングの結果をもとにアーキテクチャ定義を修正し，アーキテクチャを詳細化する

③と④は複数回繰り返して実施する．それにより，開発プロジェクトに適したアーキテクチャを定めることができる．

①〜④の結果得られる成果物は，アーキテクチャ・ベースラインおよびアーキテクチャ記述である．アーキテクチャ・ベースラインとは，最新のアーキテクチャ定義を反映した実行可能なサンプルプログラムで，これがその後の繰り返し開発のベースラインとなる．アーキテクチャ記述は，定義され実装されたアーキテクチャを解説するドキュメントである．その後の開発は，これらの成果物をもとに，システムに要求される機能を繰り返し設計・実装していくスタイルとなる．

2.3 アーキテクチャ中心アプローチを遂行するための体制

システム開発においてアーキテクチャ中心アプローチを適切に遂行するために，プロジェクトにアーキテクチャ・チームを置くことが望ましい．

アーキテクチャ・チームの責務は，開発プロジェクトに適したアーキテクチャを確立して技術リスクを解消することと，その後の繰り返し開発をスムーズに進ませることである．したがって，前節で述べた作業を中心にしながら，実装担当者に対してルールや作業手順などを準備する．

アーキテクチャ・チームは以下の成果物を生成する．

1） アーキテクチャ記述
2） アーキテクチャ・ベースライン
3） 各種ルール（コーディング/ネーミングルール，ドキュメントひな形など）
4） 各種作業手順（開発/単体テスト/リリース，仕様変更，共通処理利用など）

プロジェクトを構成するチームとして，アーキテクチャ・チーム以外に，要件チーム，プロトタイプ・チーム，構築チーム，受入チームの四つを置いた場合の例を考える．アーキテクチャ・チームを中心にチームの作業分担を俯瞰したものは図1のようになる．

図1の通り，アーキテクチャ・チームは要件チームと共に対象システムが必要とする要件を洗い出す．アーキテクチャ・チームは主として品質に対する要件を担当する．要求整理によって洗い出した要件をもとに順次リスク抽出を行う．ここまでが2.2節の①に相当する．それから，図1の「基本アーキテクチャ記述」に入る．これが2.2節の②に相当する．基本アーキテクチャが定まったら「アーキテクチャ詳細化」に入る．抽出されたリスクを考慮しながら品質に対する要件に関係が深い機能要件を選び，プロトタイプ・チームがサンプルプログラムの作成と検証を行う．これが2.2節の③に相当する．その結果に基づいて，アーキテクチャ・チー

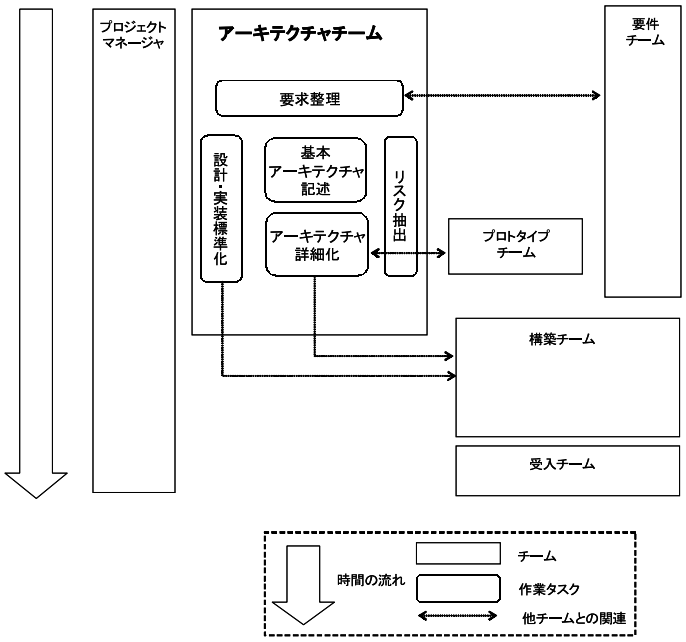

図 1　アーキテクチャ・チームを中心としたシステム開発のイメージ

ムまたは要件チームにて方針または対応を確定する．アーキテクチャの修正による対応が「アーキテクチャ詳細化」であり，2.2 節の④に相当する．アーキテクチャ詳細化の作業が完了した結果の成果物が上記の 1）および 2）であり，設計・実装標準化の作業が完了した結果の成果物が上記の 3）および 4）である．

2.4　アーキテクチャ中心アプローチの利点

アーキテクチャ中心アプローチにより以下の利点を享受できる．

技術リスクの早期顕在化とその解消

これには二つの理由がある．まず，サンプルプログラムの動作を開発の早期に検証することができる．したがって，ソフトウェアの動作を開発後期にしか確認できないウォーターフォール型の開発と比べて設計の正しさを早期に確認できる．次に，開発プロジェクトにおいて全てのサブシステムが同じ安定したシステム構造になる．したがって，サブシステムごとに発生しがちである設計のばらつきが小さくなり，全体としての整合性が保証される．

開発生産性の向上

下流工程である実装の担当者にドキュメントだけで設計の意図を伝えるには正確なドキュメントを多量に必要とする．そのようなドキュメントを作成するためには非常に多くのコストがかかる．これはウォーターフォール型開発の課題としてよく挙げられることでもある．

アーキテクチャ中心アプローチであれば，実装の担当者に設計の意図を伝える際に，ドキュメントに加えてアーキテクチャ・ベースラインとそのソースコードを利用できる．ドキュメントに記述する詳細な情報をソースコードへの参照で代用できるので，品質を保ち

ながら開発のコストを削減できる．

繰り返し型開発の容易さ

繰り返し型開発の利点は，リスクの高い部分から先に構築することでシステム開発全体のリスクを削減でき，もし手戻りが発生した場合にも影響範囲を小さくすることができることである．しかし，もしも繰り返しの単位が大きくて1サイクルにかかる期間が長ければ，短期開発がほとんどである現在のシステム開発においては繰り返しの数を多くすることが困難である．アーキテクチャ中心アプローチであれば，システムの主要な仕組みはすでに設計されているので開発1サイクルにおける作業量は小さい．そのため，システム開発を繰り返し型で無理なく実施できる．

要求リスクの早期顕在化・解消

一般に,顧客と開発者とがコンピュータ・システムの仕様を完全に共有するのは難しい．図による表記法であるUML（Unified Modeling Language）でコンピュータ・システムの仕様を表現することで厳密なコミュニケーションを図る場合もあるが，いずれにせよ顧客との間に何らかの前提知識を共有することが要請される．アーキテクチャ中心アプローチであれば，設計段階で動作する「もの」ができる．その「もの」の動作を前提知識とすれば，システムの仕様を具体的に共有できる．

3. アーキテクチャ中心アプローチの課題と解決

本章では，アーキテクチャ中心アプローチを開発プロジェクトに適用する際の課題を解決する手段であるアーキテクチャの再利用について述べ，その具体的な方法を考察する．

3.1 アーキテクチャの再利用による課題解決

アーキテクチャ中心アプローチを開発プロジェクトに適用する際の課題は，アーキテクチャの確立という作業が難しいことである．アーキテクチャ中心アプローチではアーキテクチャ記述およびアーキテクチャ・ベースラインを作成していく．しかし，共通的な基盤を設計する作業は担当者に高度なスキルとノウハウが要求される．特に，実装基盤について深い知識をもち，技術的な制約について熟知していなければならない．さらに，開発の初期にアーキテクチャ・ベースラインを開発することはスケジュール上の制約に合わない可能性がある．結果として，技術リスクを充分に解消できていない，検証が不十分なアーキテクチャができあがってしまうことになる．

この問題を解決するには，アーキテクチャの再利用が有効である．

開発プロジェクトが始まるよりも前の時点で，多くのプロジェクトで汎用的に利用できるアーキテクチャを準備できるのであれば，その後の開発プロジェクトは用意されたアーキテクチャをそのままの形かまたは部分的なカスタマイズを施した形で利用できる．準備されたアーキテクチャがよく検証されたものであればあるほど，アーキテクチャ担当者の負担が軽減されるため，アーキテクチャ・チームのスキルリスクを低減できる．また，実行可能なソフトウェアも最初から用意されているのであれば，短期間のプロジェクトであってもアーキテクチャのカスタマイズに対する検証を早い段階から実施でき，技術リスクの低減にもつながる．

建物の建築に例えれば，毎回建造物の設計思想から始める建築型のシステム開発ではなく，設計思想，土台，部品を再利用して建物を組み立てていく製造業型のシステム開発を実施する

ということである．結果として，理解しやすく実践しやすい「ライトウェイト」な開発プロセスの実施が可能になる．

3.2　アーキテクチャを再利用するには

そこで，どうやってアーキテクチャを再利用するかが問題になる．アーキテクチャ中心アプローチとはアーキテクチャの定義とそれを実現するアーキテクチャ・ベースラインの両方をシステム開発の早期に作る手法であった．では，アーキテクチャ・ベースラインを伴うアーキテクチャの再利用はどのように実施すればよいのだろうか．

最初に考えられるのは，フレームワークと呼ばれるソフトウェアを導入することだが，この方法でアーキテクチャを再利用するのは現実的ではない．フレームワークとはアプリケーションの制御アルゴリズムを再利用するためのオブジェクト指向に基づいて作られるソフトウェアであるが，フレームワークにはアプリケーション全体をカバーすることと汎用性の高さとがトレード・オフの関係になってしまうという課題があるためである．また，大規模システムであるほど制御アルゴリズムの独自性が強いため，再利用にメリットがなくむしろコスト増の原因となる．したがって，特に大規模プロジェクトにおいては，プロジェクト開始前に汎用的なフレームワークを開発しておくことにはメリットが少ないと考える．

次に，フレームワークという制限の多い形でアーキテクチャを再利用するのではなく，アーキテクチャ定義そのものの再利用に主眼を置き，実行可能プログラムの再利用に関してはアーキテクチャ定義のサンプル実装という形で事前に準備するという手法を考える．フレームワークとの一番の違いは実行可能プログラムの扱いである．フレームワークの場合はビルド済みのバイナリ部品を再利用するが，この手法ではビルド前のソースコードを部品として再利用するのである．これにより，制御アルゴリズムに独自性の強い大規模システムの開発においてもプロジェクト開始前に準備しておいたアーキテクチャを再利用することが可能になる．

当社の.NET Framework[*1] 向け開発方法「LUCINA for .NET」では，上に挙げた 2 種類の手法のうち後者を採用している．LUCINA for .NET では，再利用するアーキテクチャ定義のことをアーキテクチャの「事前定義」，再利用する実行可能ソフトウェアをアーキテクチャの「事前実装」と呼び，両者をあわせて，「アーキテクチャ・テンプレート」と呼んでいる．アーキテクチャ・テンプレートとは実際の開発プロジェクトで確立するアーキテクチャのひな形であり，アーキテクチャ・チームに対する初期入力となる．

4.　LUCINA for .NET が提供するアーキテクチャ・テンプレート

本章では，前章で述べた問題を解決するために当社の.NET Framework 向け開発方法「LUCINA for .NET」が提供している「アーキテクチャ・テンプレート」の内容とその利点について述べる．

4.1　アーキテクチャ事前定義

4.1.1　アーキテクチャ事前定義の目的

LUCINA for .NET のアーキテクチャ事前定義とは，基本アーキテクチャ記述およびアーキテクチャ詳細化のコストとリスクを最小にするために，アーキテクチャの設計方針を示すものである．当社の開発方法「LUCINA for .NET」では，「LUCINA　テクニカルドキュメント for

.NET　設計方針編」（以下，「設計方針編」）というドキュメントの形で提供される．

これを入力として，「アーキテクチャ詳細化」の作業でカスタマイズした成果物が，アーキテクチャ・チームの成果物である「アーキテクチャ記述」となる．つまり，設計方針編は，アーキテクチャ記述のひな形として使われることを想定して作成してある．

アーキテクチャ記述のひな形を用意することで，アーキテクチャ詳細化における技術リスクとコストの削減効果が期待できる．

4.1.2　アーキテクチャ・テンプレートの想定する企業システム構造

設計方針編では，最初に企業システムの構造を定義している．図2に示すのが企業システムの構造である．企業システムは「コンポーネント・システム」いう名のサブシステムとそれをつなぎ合わせる「ターゲット層」を中心に構成する．コンポーネント・システムの例としては，eコマース・システムを例にとれば「カタログ管理コンポーネント・システム」や，「販売管理コンポーネント・システム」といったものが挙げられる．それ以外の企業システムの構成要素として，ユーザー・インタフェースを担当するサブシステムである「フロントエンド」と，企業間連携に特有のロジックを局所化する「ゲートウェイ層」を置く．

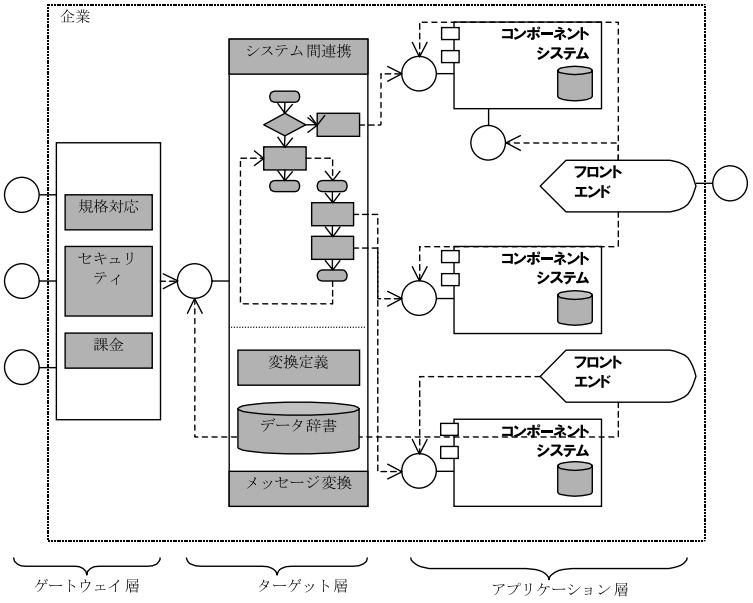

図2　企業システムの構造

設計方針編で定義される企業システムの構造は，以下の特徴を持つものである．

企業システムを，属性（業務データ）とそれに対する操作（業務ロジック）とをまとめたグループに分割し，情報隠蔽の単位としている

オブジェクト指向でいうところの「カプセル化」である．コンポーネント・システムが内部で利用するデータを直接外部から操作することを禁止し，業務データにアクセスする際はコンポーネント・システムが提供するインタフェースを利用することを強制する．

コンポーネント・システムは，フロントエンドとターゲット層に対して，業務ロジックをサービスとして提供する

ここでは，他のコンピュータ・システムによってリモート呼び出しの形で呼び出されることでロジックが実行されるものを指して「サービス」と呼んでいる．ユーザ・インタフェースに改修を加えたりユーザ・インタフェースを入れ替えたりするような変更が発生してもコンポーネント・システムに影響が及ばないように，コンポーネント・システムからユーザ・インタフェースを独立させているのである．

このような特徴をもつ「コンポーネント・システム」を企業システムのサブシステムとすることでサブシステムの独立性が高まる．したがって，企業システムは迅速かつ柔軟に構築・変更できる柔軟性を得ることができる．

コンポーネント・システムによる企業システム分割の一番の利点は，並行開発の実施が容易なことである．各コンポーネント・システム同士の独立性が高いため，コンポーネント・システムの内部を設計し実装していく作業は他のコンポーネント・システムの構築作業と独立して実行できる．したがってコンポーネント・システム単位での並行開発がスムーズに実施できる．

開発者の教育コストを低減

大規模開発においては，各コンポーネント・システムの構築はそれぞれ別の構築チームが担当することが多い．しかし，各コンポーネント・システムの構造は統一されているので，構築チームに対して行うべき教育は多くが共通している．したがって，コンポーネント・システムに分割されていないシステムの開発に比べ，開発者に対する教育のコストが抑えられる．

4.1.3　.NET Framework での実装を前提としたソフトウェア・アーキテクチャ

設計方針編で定義している企業システムの構造は，企業システムがリレーショナルデータベースを中心とした業務アプリケーションの集合となっていることが多い現実をふまえて，そのような業務アプリケーションを適切なサイズに分割し，企業システム全体を再構成するという考え方に基づいたものである．そして，LUCINA for .NET のアーキテクチャ事前定義は，以上のような考え方を踏まえ，「コンポーネント・システム」およびそれを操作するためのユーザ・インタフェースとなる「フロントエンド」について .NET Framework での実装を前提としたアーキテクチャを定義している．

コンポーネント・システムおよびフロントエンドのソフトウェア・アーキテクチャを図 3 に示す．フロントエンドはユーザ・インタフェースのデザインとロジックの分離を目指し，《Form》《Event Handler》《Model》の三つに分割している．また，コンポーネント・システムはビジネスロジック層/データアクセス層/データ層の 3 層構造をとり，《Control》《Business》《Target Proxy》《Data Access》の四つに分割している．そして，データの受け渡しの入れ物となるオブジェクトとして，《Data Transfer Object》をフロントエンドとコンポーネント・システムとで共通に使う．

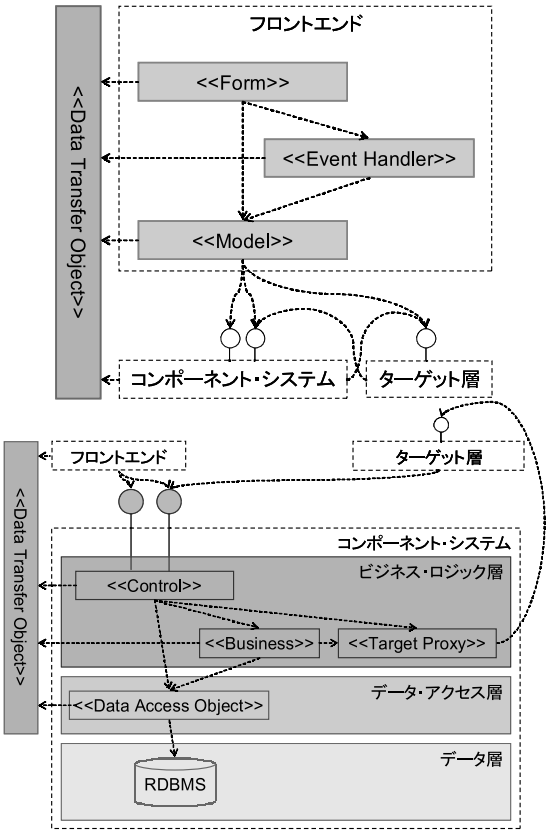

図3　コンポーネント・システムとフロントエンドのソフトウェア・アーキテクチャ

設計方針編で定義しているコンポーネント・システム，およびフロントエンドのソフトウェア・アーキテクチャは，以下のような特徴を持っている．

- .NET Framework が提供する API を素直に利用するため，Visual Studio .NET[*2] などの開発ツールとの親和性が高い．
- ソフトウェアの構造および利用する要素技術が，決定の理由も含めて詳細に記述されている．
- 物理的な配置に関して，ハードウェアの冗長化などの自由度を持たせることができるように設計されているため，スケーラビリティ，可用性に優れている．
- コンポーネント・システムの通信インタフェースは，ターゲット層の実装に合わせて XML Web サービスと COM＋プロトコルの二つを選択肢として用意している．

4.1.4　アーキテクチャ事前定義の利点

コンポーネント・システム，およびフロントエンドのソフトウェア・アーキテクチャが事前定義されていることにより以下の利点を享受できる．

開発生産性の向上

まず，アーキテクチャ定義のひな形が最初から用意されているため，アーキテクチャ中心アプローチにおいてアーキテクチャ候補の決定が単純化される．また，設計方針編で定

義しているアーキテクチャはすでに検証が済んでいるため，カスタマイズ不要な範囲においてはアーキテクチャを検証し，修正するという繰り返しの流れを省略することができる．

次に，ここで説明したようなアーキテクチャが定義されたドキュメントがあらかじめ用意されていることで，「設計・実装標準化」の作業タスクにおいてドキュメント作成に必要なコストが大幅に削減される．

また，Visual Studio .NET などの開発ツールとの親和性が高いため，構築チームの作業を効率化することもできる．

スキルリスクの低減

上で述べた特徴を見ても分かるとおり，アーキテクチャの再利用および要件に応じたカスタマイズが容易に行えるように，アーキテクチャに一定の自由度が与えられている．与えられた自由度の中での選択肢についてはすでに検証が済んでいるため，アーキテクチャ・チームの開発者は .NET Framework という実装プラットフォームから受ける技術的制約について詳細に把握する必要がなくなっている．

また，Visual Studio .NET などの開発ツールとの親和性が高いため，構築チームの作業に必要な知識を限定できる．

4.2　アーキテクチャ事前実装

4.2.1　アーキテクチャ事前実装の目的

LUCINA for .NET のアーキテクチャ事前実装とは，4.1 節で説明したアーキテクチャを実装したアプリケーションのサンプルである．当社の開発方法「LUCINA for .NET」では，「LUCINA Web Foundation for .NET」という名前の製品になっている．LUCINA Web Foundation for .NET は実際に動作するアプリケーションとして，ソースコードの形で提供される．

これを「アーキテクチャ詳細化」の作業タスクでカスタマイズした成果物が，業務分析フェーズにおけるアーキテクチャ・チームの成果物である「アーキテクチャ・ベースライン」となる．つまり，LUCINA Web Foundation for .NET はアーキテクチャ・ベースラインの「ひな形」として使われることが想定されている．

アーキテクチャ・ベースラインのひな形を用意することで，アーキテクチャ詳細化において技術リスクが小さくなりコストが削減される効果が期待できる．

4.2.2　アーキテクチャ事前実装の特徴

LUCINA Web Foundation for .NET は，アーキテクチャの事前定義に準拠した形で，e コマースサイトの Web アプリケーションおよび商品カタログ管理用 Windows アプリケーションを実装している．

図 4 に示したとおり，LUCINA Web Foundation for .NET は，四つのコンポーネント・システム（商品カタログ管理コンポーネント・システム，商品販売コンポーネント・システム，クレジットカード与信コンポーネント・システム，在庫管理コンポーネント・システム）と，二つのフロントエンド（e コマース Web フロントエンド，商品カタログ管理スマートクライアント），そしてコンポーネント・システムをつなぎ合わせるターゲット層から構成されている．

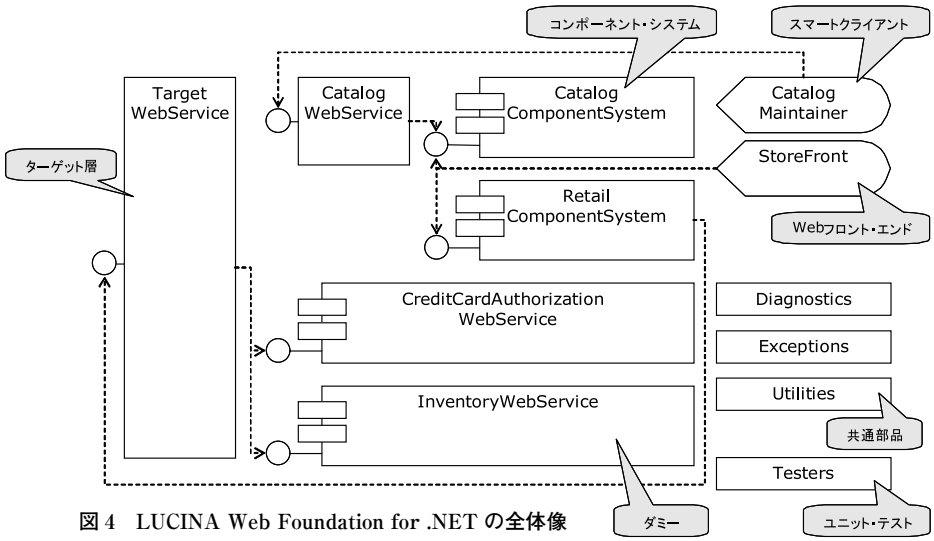

図 4　LUCINA Web Foundation for .NET の全体像

LUCINA Web Foundation for .NET で提供されているアプリケーション実装は以下の特徴を持っている．

- Visual Studio .NET のコード自動生成機能，ビルド～ソフトウェア配置～テスト迄を自動実行するツール nUnit（http://www.nunit.org/）等，ツールを活用した開発ができる．
- .NET Framework が提供している API を隠蔽せず，可能な限りそのまま使っている．
- 当社独自の製品や，第三者ベンダ製品の機能に依存していない．

4.2.3　アーキテクチャ事前実装の利点

アーキテクチャの事前実装により以下の利点を享受できる．

開発生産性の向上

アーキテクチャ・ベースラインのひな形が用意されているので，アーキテクチャの検証が早期に実施できる．その結果，アーキテクチャの確立に要する期間を短くできる．

また，構築チームは各種開発ツールによるサポートを受けながら作業できるため，効率的な開発が期待できる．

スキルリスクの低減

特殊な API や機能を使っていないため，平均的な .NET Framework 開発者であればスムーズに理解でき，開発に必要なスキルの習得が容易である．したがって，プロトタイプ・チームおよび構築チームのスキルリスクを低減させることができる．

技術リスクの低減

マイクロソフト社以外のベンダによる技術に依存していないため .NET Framework のバージョンが上がっても大きな問題なく対応できる．

4.3　アーキテクチャ・テンプレートによる効果

当社では，アーキテクチャを中心としたシステム構築のポイントを抽出・検証し，実際のシステム開発において容易に適用できるもの，すなわち LUCINA for .NET を提供している．

システム構築におけるアーキテクチャ設計・運用に，LUCINA for .NET が提供するアーキテクチャ・テンプレートを利用することで，各システム構築プロジェクトでのアーキテクチャ設計における負荷軽減と設計ミス・実装ミスを極力減らすことが可能となる．

また，アーキテクチャ・テンプレートに対応する各種ガイド・ひな形を利用することで運用面を含めた迅速な作業決定が可能となり，さらにリスク調査要件，共通処理などの部品化が容易となる．

本稿に記述した各種試みをユニシスグループにおいて継続・活用することで，以下にあげるグループ・コアコンピタンスの強化を図ることが可能と考える．

・開発工数の低減による市場競争力の向上
・成果物品質向上による手戻り等減少で実現する工期短縮と開発工数低減，顧客信頼度向上
・リスクの事前抽出による開発事故の防止で実現する不要コストの削減

5. LUCINA for .NET の拡充

本章では，LUCINA for .NET で提供される情報の拡充と精度向上を図るための取り組みについて述べる．

5.1 陳腐化の防止

システム構築において有効となる情報を提供し続ける場合には，情報内容の陳腐化を防ぎ，整理された状態での情報拡充を継続する必要がある．図 5 において LUCINA for .NET で提供される情報の概要を記述しているが，これら情報の更なる拡充および精度の向上を継続することで，より広範囲な利用と精度向上が実現する．

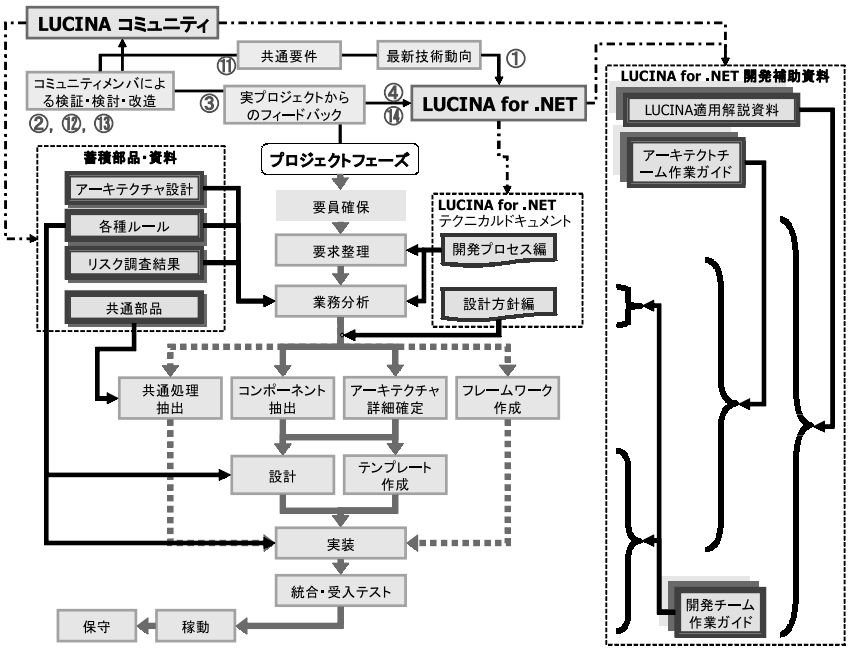

図 5　情報拡充への取り組み概要

これら情報拡充と精度向上を図るために，アーキテクチャ・テンプレートは随時 LUCINA for .NET にて提供する．各種ガイド・ひな形・部品は LUCINA コミュニティで提供することで，全社を通じた技術・情報の拡充が可能と考える．

図５は，現在弊部で実施している陳腐化防止と情報拡充への取組み概要である．『最新技術の反映』および『共通要件の蓄積』それぞれについての取組みと効果は以下となる．なお，表１ａおよび１ｂにおける丸付き数字は，図５のそれに対応している．

表１ａ　最新技術の反映

No	取組み	効　　果
①	**最新技術** の検証とLUCINA for .NET 提供コンテンツへの取込み	様々な企業システム要求に対応する最新技術の蓄積による、最適な選択肢の提供
②	**最新技術** のLUCINAコミュニティでの検証	各種基盤技術に精通したメンバの 技術検証とディスカッションによる最新技術要件の多角的な技術要素に基づいたブラッシュアップ
③	実プロジェクトでの**最新技術** 適用成果物LUCINAコミュニティへのフィードバックと検討	各種業務要件に精通したメンバの利用技術検証とディスカッションによる最新技術要件の多角的な業務要素に基づいたブラッシュアップ
④	実プロジェクトでの**最新技術** 適用成果物の、LUCINA for .NET提供コンテンツへのフィードバック	最新技術の様々な実装要件での適用実績の蓄積による、開発リスクの事前軽減

表１ｂ　共通要件の蓄積

No	取組み	効　　果
⑪	**共通要件** の検証とLUCINA for .NET提供コンテンツへの取込み	様々な企業システム要求に対応する共通要件の蓄積・活用による、生産性の向上
⑫	**共通要件** のLUCINAコミュニティでの検証	各種基盤技術に精通したメンバの技術検証とディスカッションによる共通要件の多角的な技術要素に基づいたブラッシュアップ
⑬	実プロジェクトで発生した**共通要件** 適用成果物のLUCINAコミュニティへのインプットと検証	各種基盤技術に精通したメンバの技術検証とディスカッションによる共通要件の多角的な技術要素に基づいたブラッシュアップ
⑭	実プロジェクトで発生した**共通要件** 適用成果物のLUCINA for .NET提供コンテンツ へのフィードバック	共通要件の様々な実装要件での適用実績の蓄積による、開発工数の低減

表２　.NET Framework 上でシステムを構築したシステム開発事例

顧客とシステム概要	概　　要	
金融業 受付紹会システム	構造 特徴	・XML Webサービスを DMZ-LAN間に活用した3層構造 ・MAPPERをバックエンドとしたフロントシステム用フレームワーク作成
	共通 要件	・中規模開発のWBS，フレームワーク定義書作成→サンプル化 ・リスク調査要件，共通部品，各種チェックリストの作成
サービス業 日常全業務改革	構造 特徴	・スマートクライアントを利用したリッチなクライアント環境の提供 ・EAIを活用した疎結合システム
	共通 要件	・大規模開発のWBS，アーキテクチャ記述書作成→サンプル化 ・テスト手法と効果／ドキュメンティング，AP制御(閉塞等)部品の作成 ・リスク調査要件，共通部品，各種チェックリストの作成
製造販売業 特定目的 ポータル	構造 特徴	・ポータルフロントと既存システムとのシングルサインオン ・DWHシステムとの融合
	共通 要件	・ポータル構築共通部品，リスク調査要件 ・DWHシステム（Mart Solution）の結合部品
流通業 受発注システム	構造 特徴	・DBにORACLEを採用（従来SW活用）
	共通 要件	・処理モデルの適用 ・リスク調査要件，共通部品，の作成

5.2 実システムでの適用

本稿で紹介したアーキテクチャ中心アプローチに基づくシステム開発は既に数例を数える．

いずれの案件においても，5.1 節で紹介した陳腐化の防止にかかわる情報を有効活用し，該開発案件での成果物自体が LUCINA コミュニティ，LUCINA for .NET へと蓄積されることにより，更なる生産性向上，品質向上，開発事故防止を推進している．

表 2 に.NET Framework 上でシステムを構築した実開発適用例をまとめる．

6. おわりに

本稿では，.NET Framework を基盤とした中〜大規模システム開発におけるアーキテクチャ中心アプローチによるアーキテクチャの確立プロセスと，それを可能にするアーキテクチャ・テンプレートについて紹介した．

現在，アーキテクチャ中心アプローチの適用実績を積み重ねている過程であり，その具体的効果に関する言及は，次の機会に譲ることにする．

＊1 マイクロソフト社の XML Web サービスを中心にすえた「.NET 構想」を実現するアプリケーション基盤「Microsoft® .NET Framework」.

＊2 .NET Framework に対応したマイクロソフト社の総合開発ツール「Microsoft® Visual Studio® .NET Version 2003」.

執筆者紹介 今道正博（Masahiro Imamichi）

1991 年日本ユニシス(株)入社．銀行勘定系システム開発に携わる．2002 年の .NET ビジネス専任組織「.NET ビジネスディベロプメント」で LUCINA for .NET を活用したシステム構築および EIP システムの構築およびコンサルティングに従事．

猪股健太郎（Kentaro Inomata）

2000 年日本ユニシス(株)入社．Java を利用した Web アプリケーションシステム開発に携わる．2002 年の .NET ビジネス専任組織「.NET ビジネスディベロプメント」で「LUCINA テクニカルドキュメント for .NET 設計方針編」を作成．現在 .NET テクニカルエヴァンジェリストとして講演・執筆活動に従事．