第7回 ソフトウェアアーキテクチャの今日的価値

岸 知二 NECソフトウェアデザイン研究所
友枝 敦 富士総合研究所
深澤良彰 早稲田大学理工学部情報学科

ソフトウェアアーキテクチャの研究が，近年，米国を中心に活発化している[1)~5)]．

アーキテクチャという用語は一般的に広く用いられており，ソフトウェアのアーキテクチャという言い方も以前から日常的に使われている．このためオブジェクト指向やコンポーネントウェアといった用語と違って，ソフトウェアアーキテクチャという用語は比較的自然に受け入れられているようである．逆にそれが災いしてか，「ソフトウェアを作る人は常にそのアーキテクチャを考えてきた．ソフトウェアアーキテクチャなどというが，一体どこが新しい点なのだろうか」と受け取られることもある．

本稿では，ソフトウェアアーキテクチャの研究，開発の今日的な意義とその背景，活用について述べる．

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は平たく言えばソフトウェアの構造のことである．ソフトウェアをその構成要素と構成要素間の関係として理解しようとするものである．冒頭で述べたように，従来からもソフトウェアを作る人は常にその構造を考えてきた．したがって，こうした構造に関する研究自体は新しいものではない．1960年代後半から1970年代前半，構造化プログラミングを中心としてソフトウェア工学が大きく発展を遂げた時代に，すでにDijkstraによる階層構造の有効性の指摘や，Parnasによる情報隠蔽の提唱など，ソフトウェアの構造に関する重要な提案が行われていた．またモジュール強度や結合度などの概念も提案された．

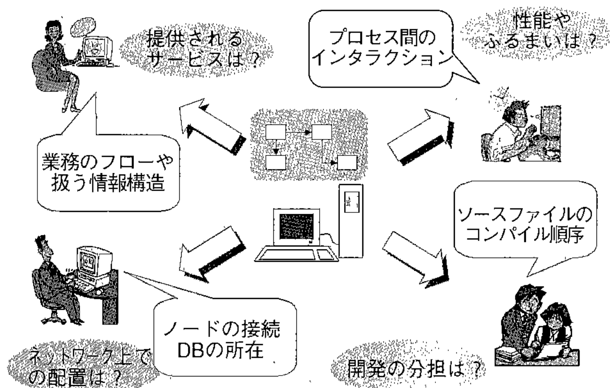

図-1 目的に応じて異なるアーキテクチャが必要

しかし，ソフトウェア開発の現場で，実際にどのような構造が議論されているかを考えてみると，ある人はOS，ミドルウェア，アプリケーションといった階層構造を議論し，別の人は実行時のコンポーネントの呼び出し関係を議論したりと，構造を議論する目的や視点，記述方法などはきわめて多様であり，共通の認識に乏しかった．

さらに，ソフトウェアアーキテクチャといっても，機能的な側面，性能，開発分担などの議論の目的に応じて，異なったビュー（視点）でとらえる必要もある（図-1）．こうした状況に対して，Kruchenはソフトウェアアーキテクチャを論ずるビューを4つに整理し，各ビューに応じたソフトウェアアーキテクチャの記述方法を提案している（表-1）[6)]．

ソフトウェアアーキテクチャを理解するためには，ソフトウェアアーキテクチャとは何かといった一般的議論ではなく，このように目的ごとに個別のソフトウェアアーキテクチャから理解する方が実用的であり分かりやすい．なお，上述したKruchenのビューはソフトウェアアーキテクチャの唯一の分類ではなく，他の分

表-1 Kruchenによるアーキテクチャの4つのビュー

ビュー	記述内容	記述要素例
論理	機能的な要求やサービス	パッケージ，クラス，関係
プロセス	並列性や分散	プロセス，メッセージ
配置	ハードウェアへのソフトウェアの配置	ノード，コネクション
コンポーネント	開発での静的構成	パッケージ，依存関係

類も提案されている．

どのようなアプリケーションでも何らかの構造を持っているが，再利用などの議論をする際にはそうした特定のソフトウェアの構造だけでなく，複数のソフトウェアに繰り返し用いられるアーキテクチャを再利用したいことも多い．この両者を区別するために，特に後者をアーキテクチャスタイルあるいはアーキテクチャパターンと呼ぶ．

ソフトウェアアーキテクチャの今日的価値の発見

ソフトウェアの研究・開発の変化によってソフトウェアアーキテクチャの価値が見直され，その重要性が認識されるようになった(図-2)．

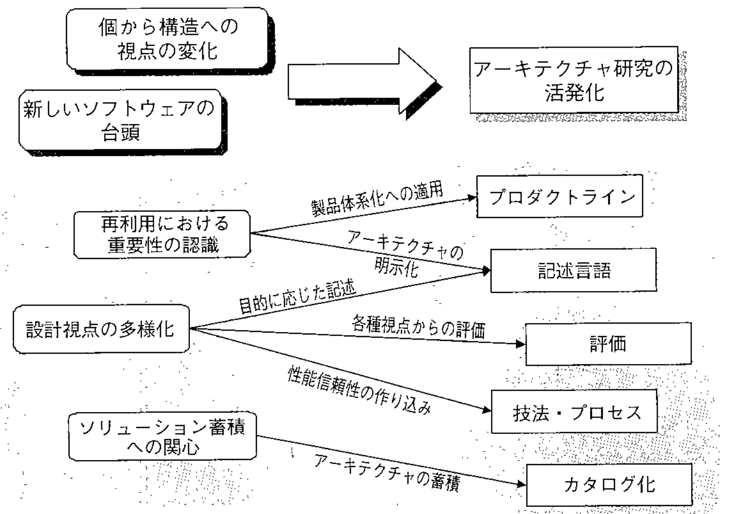

図-2　アーキテクチャ研究とその背景

• 個から構造への視点：構造の明示的な表現と相互理解：大規模なシステムになるほど，その構造を正しく捉えたり開発者相互で理解を共有することが困難となる．ソフトウェアアーキテクチャは，ソフトウェア全体の構造を表現し，設計できるようにすることによって，より大規模で複雑なソフトウェアの設計を可能とする方法である．関数，クラス，コンポーネントなどのソフトウェアの個々の構成要素から，それらが他の構成要素とどのような関係を持ち，どのような役割を持つかといった構成要素間の関係やソフトウェア全体の構造の重要性が認識されるようになった．

• 非機能的特性との関係：ソフトウェア開発においては機能とともに，性能，信頼性，再利用性といった非機能的特性の実現も重要である．この非機能的特性はソフトウェアアーキテクチャと強い関係を持っている．たとえば，性能を左右するコンポーネント間のインタラクションの回数は機能の割振りや制御構造によって特性づけられ，信頼性は二重化などの構造によって影響される．再利用性を高めるためには，変更の行われやすい部分と行われにくい部分とを構造的に分離することが必要となる．

• 設計視点の多様化：ソフトウェアの複雑度が高まるにつれて，多様なビューから設計できる必要がうまれた．構造化技法においては，設計の主要な視点は機能のみであった．また，情報システムの分析の主要な視点は，E-R図などで表される情報やデータ要素とその間の関係である．オブジェクト指向技法では，ソフトウェアを静的な構造や動的な振舞いなどの多面的な視点で捉え，修正や拡張性が意識されるようになった．ソフトウェアアーキテクチャでは，さらに，多様な非機能的特性もその視野に含めることができる．アーキテクチャや設計の視点は，ソフトウェアの黎明期から議論されてきたテーマであるが，その内容は多様化，高度化している．今では，多様な視点からアーキテクチャや設計を議論することが求められるようになった．

• 再利用：ソフトウェア部品の再利用を妨げる一因として，アーキテクチャの不整合（architectural mismatch）の問題が指摘された[6]．部品はそれがどのように呼び出されるか，どのようなデータ構造を扱うかなど，何らかのアーキテクチャを想定して作られている．こうしたアーキテクチャ上の想定が，部品とそれを利用する側とで整合しなければ，再利用は難しい．ラッパーなどの変換が必要になる．こうしたアーキテクチャ不整合の問題は，再利用部品の粒度が大きくなるほど制御が大掛かりとなるので，問題が顕在化する．フレームワークやコンポーネントウェアのような粒度の大きな再利用部品は，利用する側の制御構造やデータ構造を強く規定する．アーキテクチャ整合性は再利用の鍵となる．

• 具体的なソリューションの蓄積：デザインパターンやアーキテクチャパターンは，実際のソフトウェア開発において積み重ねられてきたアーキテクチャや局部的な設計のノウハウをパターン化したものである．利用しやすいように，設計課題とその設計例を対応づけて整理・体系化している．これは，従来の設計技法が，考え方の枠組みは提示していたものの，解決策はその技法を利用する設計者がそのつど考えなければならなかった段階から大きく前進した．このように，個人のノウハウを共有できる知識として蓄積することに，研究の関心が広がってきている．

• 新しいソフトウェアの台頭：今，新たにソフトウェアアーキテクチャを議論し，アーキテクチャパターンを蓄積しなければならない背景の1つは，ソフトウェアそのものが大きく変わっているからである．特にネットワーク社会の進展はソフトウェアアーキテクチャに大

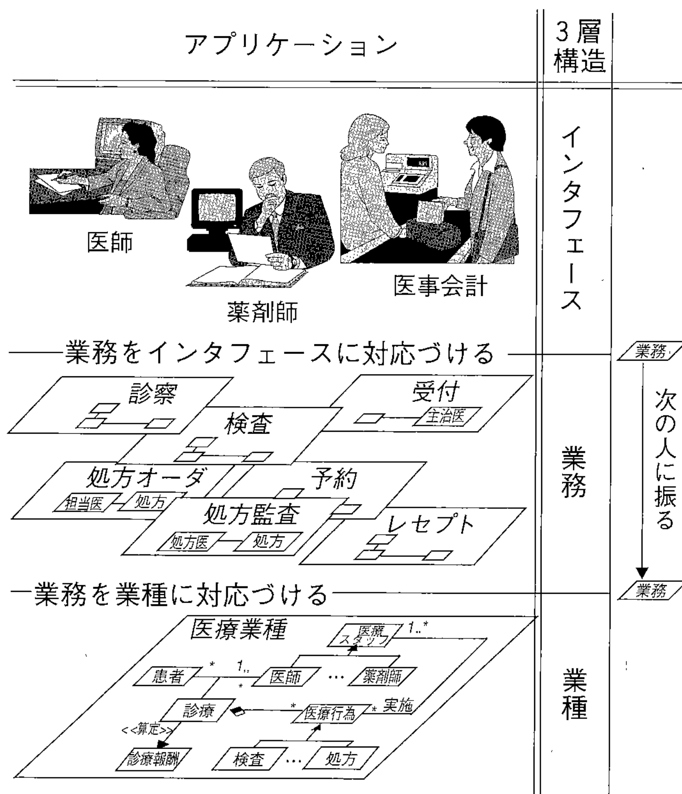

図-3　事例のアーキテクチャ

きなインパクトをもたらしている．

従来はソフトウェアアーキテクチャはアプリケーションを開発するベンダー内での問題であったが，WWWやCORBA（Common Object Request Broker Architecture）などのアーキテクチャは複数ベンダー間で共有されて初めて意味を持つ．複数のベンダーがWWW上でのビジネスシステムのアーキテクチャを提唱しているが，これはアーキテクチャを制することの重要性への認識の高まりといえる．現実の開発現場においてもソフトウェアアーキテクチャが明らかに意識される時代になっている．

アーキテクチャ研究のもたらすもの

• アーキテクチャ記述言語：ソフトウェアアーキテクチャを記述できるためには処理のアルゴリズムやデータ構造ではなく，目的に応じたソフトウェアの構造を表現できる必要がある．また，こうして記述されたソフトウェアアーキテクチャによって，ソフトウェアの性能の検討や開発分担の議論ができる必要がある．このため，アーキテクチャ記述言語ADL（Architecture Description Language）が種々提案されている．ADLは，一般に，構成要素（コンポーネント）とその間の関係（コネクタ）を記述できる言語でアーキテクチャを記述する．さらに，ADL記述に基づいて，シミュレーションや形式的な意味づけなどにより，さまざまな解析ができる．一般にADLは特定の目的に焦点を絞ったものである．たとえばRapideはイベントに対する振舞いを定義する言語であり，シミュレーションにより動的な側面からアーキテクチャをプロトタイピングできる．

• アーキテクチャの評価：アーキテクチャ評価技法の研究も行われている．SAAM（Software Architecture Analysis Method）は複数のアーキテクチャを定性的な相対評価によって比較する技法である．この技法では，コンポーネントとコネクタを基本要素とするADLを用いてアーキテクチャを記述し，それらをシステムが遭遇する重要な局面の記述（シナリオと呼ばれる）に照らして評価する．たとえば，システムに特定の変更を加えたときに影響を受けるコンポーネントの範囲などの評価によってアーキテクチャを評価する．

• 開発技法・プロセス：ソフトウェアアーキテクチャを活用して性能や信頼性などの非機能的特性を検討できることが設計の重要な課題である．従来，アーキテクチャが原因の性能問題はシステムが統合される開発末期にならないと顕在化しづらかったため，アーキテクチャの修正は重大な手戻りとなるリスクがあった．Royceは，そうしたアーキテクチャ上のリスク軽減の視点から，ソフトウェアの繰返し（iterative）開発プロセスを整理している．また，Buhrはユースケースマップと呼ばれるアーキテクチャの動的振舞いを表現できる図を活用した開発技法を提言している．

• カタログ化：デザインパターン同様に，アーキテクチャ設計上のノウハウを共有するために，アーキテクチャパターンをカタログとして蓄積するアプローチもBuschmannなどが試みている．

• プロダクトライン：類似した複数のソフトウェア製品をプロダクトラインと呼ぶ．プロダクトライン内の開発では，再利用部品を体系立って管理し効率的な開発を行うことが期待される．そのためにはプロダクトライン内の製品の整理，開発・実行環境の統合，部品の構成管理などが必要となる．特に，製品のベースとしてのソフトウェアアーキテクチャの整備が鍵となる．Celsius社では軍事関係のソフトウェア製品を体系化することで成果をあげているが，その中でもアーキテクチャの重要性が強調されている．プロダクトライン内の再利用はアーキテクチャのみで達成できるものではないが，アーキテクチャがきわめて重要かつ中心的な役割を果たしている．

アーキテクチャを中心にした分析・設計事例

アーキテクチャを中心に分析・設計を行った事例を紹介する[7]．対象は大規模病院における医療業務支援である．

• <u>基本方針</u>：分析・設計においては，システムの性能要件より，制度や要求の変更に対応できる柔軟性と分析・設計成果の再利用性の向上に重点を置いた．このため，個別要求や既存システムを出発点とするよりも，医療ドメインの調査分析を行い，これをもとにソフトウェアのアーキテクチャを決定し，開発を行うこととした．

• <u>医療ドメインの分析</u>：医療活動を，医療という業種全般に関するものと，医療活動における種々の業務に関するものの2種類に分類して抽出した．前者は医業や診療を表すドメインモデルであり，主に構造を表す静的モデルとシナリオでモデル化した．後者は医療システムの活動単位を表すものであり，たとえば，受付，診察，オーダリング，予約，検査，処方監査，レセプト発行などがあげられる．各業務ごとに，静的，動的，機能の3つの視点からモデル化した．業務の中には医療分野以外でも利用できる可能性のあるものもあるので，できるだけ一般的な業務の特殊形として取り扱えるよう留意した．このように，まず，ドメインモデルを確立することが重要である．

• <u>ソフトウェアアーキテクチャの概要</u>：ドメインモデルを実現するソフトウェアアーキテクチャをドメイン指向ソフトウェアアーキテクチャと呼ぶ．本事例では，医療共通な構造の上に各種業務群を配し，各業務にインタフェースを対応づけた3階層アーキテクチャとした．この概要を図-3に示す．

最下層が，医療ドメイン分析の結果得られた医療業務の構造を表す層であり，アプリケーション全体のプロセスの構造を表す．たとえば，医療という業種においては，いつ，誰が，どのような処置や処理を行ったのかを記録，参照できることが重要である．これらの情報は業種層において管理される．事務効率の向上などを目的として業務の流れを変更する場合には，この層の変更が必要である．

中間の業務層において，各業務の内容と業務間の関連を表現する．各業務間をつなぎ，システムとしての連携をとる機構として，依頼者／依頼部署と実施部署／実施者間のプロトコルを設定した依頼実施機構を用意した．これにより，各業務を具体化する際には，ホワイトボックスとして取り扱うが，業務をつなぐ依頼実施機構への組み込み時には，ブラックボックスコンポーネントとして取り扱うことが可能になる．個々の業務を独立に交換できるので，性能要件などにより，同じ仕様の業務を実現する異なる実装に置き換えることが可能である．また，業務開発時には，並列開発が可能となる．

図の上層はユーザインタフェースである．業務層と分離することにより，各業務のインタフェースを任意のものに取り換えることが可能になる．

• <u>評価</u>：本件はアーキテクチャを中心にした分析設計を試行したところであり，今のところ実装には至っていない．本件を通じ，アーキテクチャを中心に分析・設計することでシステムの仕様変更に対する柔軟性が確保できるという感触を得ることができた．これは，医療業種共通構造，各医療業務およびインタフェースの3層に分離したこと，また，業務間を統一して連携できるようにしたことが大きいと考えられる．分析・設計結果をもとに，病院内の各医療従事者が用いるシステムとして，医療シナリオに沿いどのような挙動をし，各場面でどのようにシステムを使うか，そして各医療従事者間の業務がどのように連携がとられるかを利用者に示した．さらに，どのような変更が可能であるかを明らかにした．このシステムは従来の医療システムとは異なる視点で作られてはいたが，病院の情報システム部門に提示したところ，仕様と柔軟性の点で概ね好評を得た．ドメインに関連したアーキテクチャの良さが，システムの柔軟性や再利用性の高さ，さらにシステム提案の確かさに大きな影響を及ぼす．本件ではドメインの専門家であると同時にオブジェクト指向の研究者でもある協力者が得られた．一般的にこのような協力者は得難く，このような人材の育成が重要な課題である．

まとめ

ソフトウェアシステムが大規模化し，複雑化するにつれて，システム全体の構造をどのように設計し，実現するかが，どのようなアルゴリズムやデータ構造を使うかより重要になってきている．「クライアント／サーバ型」や「分散型」といった表現そのものがソフトウェアアーキテクチャを意識したものであり，このような用語が頻用されること自体が，アーキテクチャが重要になっていることを物語っている．

従来のように，場当たり的にアーキテクチャを設計するのではなく，優れたアーキテクチャを設計者が共有し，再利用しながら，ソフトウェアを構築していくことが重要である．このために，具体的なアーキテクチャを収集し，共有していくための技術が，今日的なソフトウェアアーキテクチャの研究と実用化の意義である．

参考文献

1) Shaw, M. and Garlan, D. : Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall (1996).
2) Bass, L., Clements, P. and Kazman, R.: Software Architecture in Practice, Addison-Wesley (1998).
3) 岸　知二: ソフトウェアアーキテクチャへの招待, 深澤, 青山編: ソフトウェア工学の基礎IV, 近代科学社 (1997).
4) Rational Inc.: Managing Successful Interactive Development Projects : A Seminar on Software Best Practices, Version 2.3 (1997).
5) Buschmann, F. et.al.: Pattern-Oriented Software Architecture, A System of Patterns, John Wiley & Sons (1996).
6) Kruchen, P.B. : The 4+1 View Model of Architecture, IEEE Software, pp.42-50 (Nov. 1995).
7) 友枝　敦他: アーキテクチャ指向による仕様変更に強い医療情報モデル作りへの取組み, 第17回医療情報学連合大会 (1997).

（平成10年8月18日受付）