ビジネスと情報技術の視点より考察した アプリケーション・モデリングと ソフトウェア・アーキテクチャに関する研究*

森澤 好臣

内容梗概

歴史的に見て、最良の技術が必ずしも市場に受け入れられていないことが多く散見される。コンピュータの世界もその例外でない。それどころか、他の業界の事例以上に、技術よりビジネスの視点が重視されている。ソフトウェア・アーキテクチャの分野では、特に厳しい覇権争いが情報技術の進歩を先取りしながら続けられている。

本論文では、まず、ビジネスの視点では日の目を見ていないが、1980 年代前半に活発に研究され、技術の視点では強い印象を研究者に与えた論理型言語による事務処理問題のアプリケーション・モデリングを提案する。このモデルの意義は、論理型言語が事務処理アプリケーションの仕様記述に使用できることを示したことである。

次に、1980 年代後半から始まったネットワーク・レイヤーからオペレーティング・システムそしてアプリケーション・システムの実行基盤であるミドルウェア・レイヤーに至るソフトウェア・アーキテクチャの覇権争いを概観しながら、筆者が参画したアプリケーション・システムの実行基盤のソフトウェア・アーキテクチャである UA(Unisys Architecture)の歴史的位置付け、筆者らが開発したフレームワークであるオープン・ソリューション・フレームワークの開発背景と概

＊奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文、NAIST-IS-DT9861022, 2002 年 1 月 29 日

要を概観する。クライアント／サーバ型のアプリケーション・システムの開発時に使用するクライアント／サーバ・ソリューション・モデルとソフトウェア・プロダクトを推奨するプロダクトセットを提案する。この適用例の一つとして、新財務システムの情報基盤を決定するためのコンサルティング・ビジネスの適用事例を紹介し、その技術的意義を論じる。

次に、オープン・ソリューション・フレームワークで導入したクライアント／サーバ・ソリューション・モデルを、ドック・イヤーと言われる情報技術の進展に対応させて、分散処理システムのアーキテクチャ・スタイルとして提案する。アーキテクチャ・スタイルは、分散処理システムをデータ領域の場所とクライアントとサーバ間の処理の形態をベースにして９種類に分類している。この９種類のアーキテクチャ・スタイルにより、分散処理型のアプリケーション・システムを開発するときに適切なアーキテクチャ・スタイルを選択する簡潔で実践的な手法を提案する。技術の視点からだけでなくビジネスの視点からも、アーキテクチャ・スタイルを特性化し、計測尺度として「サイズ」と「距離」を導入し、アプリケーション・システムの要件を表現した特性図とアーキテクチャ・スタイルの特性図間の類似性が有用であることを示した。この尺度と特性図を利用してアプリケーション・システムに適するアーキテクチャ・スタイルを選択する手法を提案する。この選択手法の適用性を確かめるために実稼働中のお客様のアプリケーション・システムに適用し、この選択手法が実践的に使用できるとの評価を得た。

キーワード

アーキテクチャ・スタイル、DCG(Definite Clause Grammar)、分散処理モデル、分散処理システム、論理型言語、Prolog、プロダクトセット、ソフトウェア・アーキテクチャ、ソフトウェア・フレームワーク

Research on Application Modeling and Software Architecture Considering from the Viewpoints of Business and Information Technology*

Yoshitomi Morisawa

Abstract

Historically, many technologies were not accepted in the market even if they are the best technology. The computer world is not the exception. In the computer world, a viewpoint of business is regarded as important a viewpoint of technology more than examples of the other industries. In the area of the software architecture, severe hegemony fights are particularly continuing to receive progress of information technology in advance.

At first, in this thesis, we propose application modeling of a business application using the logic programming language actively reported in the first half of 1980's. This modeling gave a strong impression to researchers technically, but was not recognized by the world with a viewpoint of business. The significance of this modeling shows the ability to use the logic programming language for a requirement specification of a business processing application.

Next, we provide an overview of the hegemony fights of the software architecture that have started from the late of 1980s in network layer, operating system layer through the middleware layer of the operational infrastructure of application systems. We explain the historical positioning of the Unisys Architecture which was participated

*Doctor's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT981022, January 29, 2002

by the author and is the software architecture of the operational infrastructure of application systems. We explain the background and an overview of the Open Solution Framework (OSFW) which was developed by the authors and then propose the Client/Server Solution model and the actual recommended product sets which are used in the development of the Client/Server type of application systems. For one of applications of OSFW, we explain our customer's example to decide the infrastructure of new accounting system in our consulting business and discuss its technical significance.

Next, we propose architectural styles of distributed processing systems as the enhanced Client/Server Solution model that was introduced in the OSFW corresponding to the dog-years progress of information technologies. The architectural style classifies the architecture for distributed processing systems into nine categories based on the location of data storage and the type of processing between a client and a server. We propose a simple but practical method to select an appropriate architectural style for developing an application system from the nine architectural styles. We introduce the characterization of architectural styles, measurement criteria of "distance" and "size" and show the applicability to select the conformity between a characteristic chart representing requirements of an application system and characteristic charts of architectural styles, from not only technical viewpoints but also a viewpoint of business. We propose a method to select an appropriate architectural style for an application system using these criteria and characteristic charts. We applied this selection method in concrete real application systems of our customers for getting adaptability of the method and got the positive evaluation to use a practical selection method.

Keywords:

Architectural Style, DCG(Definite Clause Grammar), Distributed Computing Model, Distributed Computing System, Logic Programming, Prolog, Product Sets, Software Architecture, Software Framework

関連発表論文

学術論文誌

1. Koji Torii, Yuji Sugiyama, Mamoru Fujii, Tadao Kasami and Yoshitomi Morisawa: Logical Programming for the Telegram Analysis Problem, *Computer Languages*, Vol.12, No.1, pp.9-20, January 1987.
2. Yoshitomi Morisawa, Katsuro Inoue and Koji Torii: Architectural Styles for Distributed Processing Systems and Practical Selection Method, Submitted to *Information and Software Technology*.

国際会議

1. Koji Torii, Yoshitomi Morisawa, Yuji Sugiyama and Tadao Kasami: Functional Programming and Logical Programming for the Telegram Analysis Problem, *Proceedings of the 7th International Conference on Software Engineering*, pp.463-472, March 1984.
2. Yoshitomi Morisawa, Hisashi Okada, Hiromichi Iwata and Haruo Toyama: A Computing Model for Distributed Processing Systems and Its Application, *Proceeding of 1998 Asia Pacific Software Engineering Conference*, pp.314-321, December 1998.
3. Yoshitomi Morisawa: A Computing Model of Product Lines for Distributed Processing Systems, Its Product Sets, and Its Applications, *Proceedings of the First Software Product Lines Conference (SPLC1)*, pp.371-394, August 2000.
   This proceedings is published as; Patrick Donohoe, SOFTWARE PRODUCT LINES – Experience and Research Directions, *Kluwer Academic Publishers*, 2000.
4. Yoshitomi Morisawa and Koji Torii: A Practical Method to Select an Architectural Style of Product Lines for Distributed Systems, *Proceedings of the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*, pp.1391-1397, Las Vegas, Nevada, USA, June 2001.
5. Yoshitomi Morisawa and Koji Torii: An Architectural Style of Product Lines for Distributed Processing Systems, and Practical Selection Method, *Proceedings of*

Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9), pp.11-20, Vienna Austria, September 10-14 2001.

This paper is also published as; Yoshitomi Morisawa and Koji Torii: An Architectural Style of Product Lines for Distributed Processing Systems, and Practical Selection Method, *ACM Software Engineering Notes*, Vol.26, No.5, pp.11-20, September 2001.

シンポジウム・研究会

1. 加藤潤三，染谷誠，板倉教，森澤好臣，山崎利治：仕様記述法の味見，*情報処理学会ソフトウェア工学研究報告*，85-SE-40，pp.67-72, 1985.2.
2. 鳥居宏次，嵩忠雄，杉山裕二，森澤好臣：Logical Programming for the Telegram Analysis Problem，*情報処理学会ソフトウェア工学研究報告*，86-SE-46，pp.41-48, 1986.2.
3. 森澤好臣，岩田裕道，外山晴夫：分散処理システムの処理モデルの一提案，*情報処理学会ソフトウェア工学研究報告*，96-SE-109，pp.17-24, 1996.5.
4. 森澤好臣：アーキテクチャとマーキテクチャの挟間で，*情報処理学会ソフトウェア工学研究会 ウインターワークショップ・イン・恵那論文集*，pp.33-34, 1998.10.
5. 森澤好臣，鳥居宏次：分散処理システムのアーキテクチャ・スタイル，*情報処理学会ソフトウェア工学研究報告*，99-SE-122，pp. 9-16, 1999.3.

全国大会

1. 森澤好臣，鳥居宏次：仕様から Prolog プログラムを作成する一手法，*情報処理学会第 27 回全国大会*，3C-8，pp.507-508, 1983.10.

解説記事

1. 森澤好臣：構文と意味の記述によるプログラムの作成，*情報処理*，Vol.25, No.11, pp.1255-1260, 1984.11.

訳書

1. 岩田裕道,森澤好臣,加藤良治,共訳:多層型クライアント／サーバ・コンピューティング　技術展望と活用アプローチ，*日刊工業新聞社*，P.460，　1998.1.
原典：D.T.Dewire: Second-Generation Client/server Computing, *McGraw-Hill*, 1997.

その他

1. Koji Torii and Yoshitomi Morisawa: Analysis of the Telegram Analysis Programs, *Bul. Electrotech. Lab*, Vol.48, No.3, pp.117-139, 1984.
2. 森澤好臣，岩田裕道，外山晴夫：クライアント／サーバ・システム構築のためのオープン・ソリューション・フレームワーク，*Unisys 技報*，Vol.16，No.2，pp.15-33, 1996.8.

目次

図目次

表目次

NAIST-IS-DT9861022

博士論文

ビジネスと情報技術の視点より考察した アプリケーション・モデリングと ソフトウェア・アーキテクチャに関する研究

森澤 好臣

2002 年 1 月 29 日

奈良先端科学技術大学院大学

情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士(工学)授与の要件として提出した博士論文である。

森澤 好臣

審査委員：　松本 健一 教授
植村 俊亮 教授
関　浩之 教授
井上 克郎 教授（大阪大学）

1. はじめに

歴史的に見て、最良の技術が必ずしも市場に受け入れられていないことが多く散見される。家電製品の世界でのβ方式とVHS方式の戦争がその代表例として語られている。製品技術の優位性より、マーケティングの重要性が語られた例である。このことはコンピュータの世界に於いても例外ではない。いや他の業界の事例以上に、コンピュータの世界ではマーケティング戦略が重要視されている。古くは、メインフレームでのIBM社とBUNCH（バロース、ユニバック、NCR、CDC、ハニウェル）と総称されたメーカおよび国産メーカの攻防、MS-DOSとCP/Mの覇権争い、WindowsとOS/2の覇権争い、Unixの世界内での覇権争い、等がある。最近では、Linuxに代表されるオープン・ソースのソフトウェアの台頭によるワークステーションでの新たな覇権争い、そして次世代のインターネット機器での攻防戦である。

本論文では、ビジネスと情報技術の視点より考察したアプリケーション・モデリングとソフトウェア・アーキテクチャを論じる。

1.1 アプリケーション・モデリング手法

アプリケーション・モデリング手法として、複合設計、階層型、データ構造型、オブジェクト指向型、論理型、関数型等々と数々の手法が開発されている。1980年代の前半に活発に議論され、その幾つかが生き残り、ソフトウェア開発現場で使用されている。現在の主流は、オブジェクト指向である。

本論文では、1980年代に技術的に注目を浴びたが、ビジネスの視点ではマイナーな方法論であるDCG(Definite Clause Grammar)に光を当てて解説し、その意義を分析する。DCGに注目するのは、入出力データをBNF(Backus-Naur Form)で記述し、Prologに変換する手法だからである。21世紀に入り、アプリケーション間やシステム間で流れるデータの表現としてXML(eXtensible Markup Language)準拠が標準になろうとしている。ビジネスの視点からも、情報技術の視点からもDCGによるアプリケーション・モデリング手法がXMLと組合わさることによって新たな展開が開けると予想される。

1.2 ソフトウェア・アーキテクチャの歴史

過去、コンピュータメーカは、自社の製品を数多く売るために、先進的な技術を開発することと並行して、自社製品の優位性をユーザに売り込んだり、自社ユーザに取り込むために、アプリケーション・アーキテクチャ、ソフトウェア・アーキテクチャやフレームワー

クと言った自社製品中心のソフトウェア体系を開発している。これらのソフトウェア・アーキテクチャを考察するとき、情報技術の視点とビジネスの視点より、その発表の裏に秘められたコンピュータメーカの真の目的を見抜く必要がある。歴史的に見ると必ずしも最良のアーキテクチャが普及したのでなく、普及させるためにマーケティング活動が重要であったことを多くの事例が物語っている。現在もこのことは、各社のフレームワークに対して言えることである。

筆者は、本論文で対象としているソフトウェア・アーキテクチャを、マーキテクチャ(Marchitecture)と呼んでいる。この言葉は、米国ユニシス社の同僚達が使用していた俗語(Slang)で、マーケティング・アーキテクチャ(Marketing Architecture)の略称である。ソフトウェア・アーキテクチャと称すると純粋な技術的なイメージが醸し出されているが、アーキテクチャの作成には、純粋な技術面のみでなく、各社のビジネスを発展させるためのマーケティング戦略が鎧の裏に隠されていることを忘れてはならない。[42]

一般論で言うと、ソフトウェア・アーキテクチャの歴史は、標準化とプロプラエタリ化の繰り返しである。まず、新しい技術がプロプラエタリとして登場し、マーケティング活動を通して市場に認知されて業界標準（世界標準という用語が最近頻繁に使われている）そして国際標準へと成熟している。そして、次の代替えの技術や新たな技術が登場し、プロプラエタリから業界標準へと再び成熟している。ソフトウェア・アーキテクチャがカバーする範囲も、アプリケーション・アーキテクチャからソフトウェア・アーキテクチャへ、そしてソフトウェア・フレームワークへと拡大の一歩を辿っている。歴史的に、ネットワーク・プロトコルの体系化から、OS の提供する機能の体系化へ、次に OS とアプリケーション間で各種の機能を提供するミドルウェアの体系化、そしてアプリケーションの体系化へ、さらにサービスの体系化へと、標準化の焦点が言い換えるとマーケティング戦略の焦点がアプリケーションやサービスの利用者の視点へと近づいてきている。

ネットワークのレイヤーは、SNA 等の各社のプロプラエタリなプロトコルから、国際標準である OSI に移行しようとしたが、インターネットの普及で業界標準であった TCP/IP が市場を席巻し、国際標準に昇格している。

OS のレイヤーは、プロプラエタリの OS 360/370 がメインフレームの世界を席巻している。Unix の世界では Posix が国際標準として規定されている。ユーザに近い PC の世界では、Windows 98/ME と WindowsNT/2000 が事実の業界標準であり、マイクロソフトは世界標準と標榜している。この世界を打破すべく Linux が事実上の業界標準として登場しようとしているが、ビジネス的に見て、Windows を超えることが出きるか否か、ここしばらく目が離せない状況である。

ミドルウェアのレイヤーは、当初各社のプロプラエタリな機能が散見されていたが、OSIのアプリケーション層で一部機能が標準化され、業界及びユーザの任意団体であるOMGにより、分散オブジェクトの基盤としてCORBAが業界標準として規定されている。その機能範囲も年々拡大している。これに対して、マイクロソフト社がプロプラエタリな分散オブジェクト基盤としてCOM+を作成していたが、2000年に入ってはインターネット時代のアプリケーションのインフラストラクチャとすべく.NET Frameworkを発表している。

アプリケーションのレイヤーは、Monolithicから Homogeneousへ、そして Heterogeneousへと構成形態が進歩し、アプリケーションの部品化の動きが始まっているが、いまだプロプラエタリな領域を脱していない。アプリケーションのモデル化やビジネス・プロセスのモデル化が活発に研究され、報告がなされている。2000年に入ってからの注目すべき動きは、Webサービスである[2]。インターネット時代のアプリケーションのあり方の一つとして活発な研究とビジネス化が開始されている。

1.3 アプリケーション処理形態のモデル化

1994～95年頃になると、日本に於いても、オープンというキーワードのマーケティングが本格的に始まった。本格的なオープン時代において、各社の各種のソフトウェア・プロダクトを使用して要求されるアプリケーション・プログラムを構築するための何らかのシステム開発指針を創出することが必要になり、筆者らは、1995年10月にオープン・ソリューション・フレームワークを作成し、日本ユニシス社内で試用後、翌年に社外発表した。このフレームワークは、メインフレーム時代からUnix時代への突入のための日本ユニシス社のメインフレームをオープン市場に向けるためのマーケティング戦略であった。

作成したフレームワークは、ビジネスの視点と情報技術の視点より企業情報システムを考察し、企業情報システムをモデル化し、情報技術基盤の情報技術を整理・体系化し、オープン時代のアプリケーション開発を支援するために各種ソフトウェア・プロダクトの最適な組み合わせとしてプロダクトセットを設定した。そのプロダクトセットの選択を支援するためにアプリケーション処理モデルを創出している。作成した処理モデルは、アプリケーション開発工程に於ける参照モデルとして、及び開発ツールの選択基準として使用された。また、アプリケーションの実行基盤（インフラストラクチャ）の選択を支援するためにもコンサルティングの道具の一つとして使用している。

このオープン・ソリューション・フレームワークで作成した七つの処理モデルは、Dog Yearsと呼ばれるほど進化の激しい情報技術の進化に追随するために、九つのアーキテクチャ・スタイルへと発展させた。

1.4 アーキテクチャ・スタイルの選択手法

開発しようとするアプリケーション・システムの最適なアーキテクチャ・スタイルを選択する手法を提案した。９つに分類されたアーキテクチャ・スタイルの様に、絶対的評価基準の無い世界での選択手法として特性の類似を利用した。分散処理システムを特徴付ける特性を抽出し、各アーキテクチャ・スタイルの特性をレーダチャートで表現し、開発するアプリケーション・システムの特性をレーダチャートで表現し、そのレーダチャート間の類似性よりアーキテクチャ・スタイルが選択できることを示した。

1.5 論文の概要

本論文は、第２章で論理型言語によるアプリケーション・モデリングを論じる。第３章でコンピュータ業界でのソフトウェア・アーキテクチャの発展の経緯を概観し、今後の方向性を論じる。第４章で、マーキテクチャの例として、筆者が創出したオープン・ソリューション・フレームワークを説明する。オープン・ソリューション・フレームワークで作成した七つの分散処理モデルは、後に九つのアーキテクチャ・スタイルに発展させた。第５章では、このアーキテクチャ・スタイルを解説し、アプリケーション・システムのアーキテクチャ・スタイル選択を支援するために、利用者や開発者及び管理者の視点よりアーキテクチャ・スタイルの特性軸を定め、計測の尺度として「距離」と「サイズ」を導入し、特性図の類似性が有用であることも示した。この尺度と特性図を利用してアプリケーション・システムに適するアーキテクチャ・スタイルを選択する手法を提案し、実アプリケーション・システムでその適応性を実証した。最後に、本論文の全体の纏めと考察を行う。

1.6 本論文で使用する用語

本章では、現場で曖昧に使用されている用語や、時代とともにその意味が変化している用語の定義を与える。

1) アプリケーション・アーキテクチャ(Application Architecture)

アプリケーションとシステムが提供するサービス（オペレーティング・システム、プログラミング言語、データベースやデータ通信など）とのインタフェースを規定したもの。

1980 年代後半の IBM 社の SAA(Systems Application Architecture)がその代表である。

2) アーキテクチャ(Architecture)

プログラムから見たシステムの属性、すなわちデータの流れ、制御法、および概念的構造機能動作のこと。(IBM S/360 を設計した J.Amdahl 氏のハードウェア・アーキテクチャ

的な見方の定義)

アーキテクチャとは、英語で建築やその設計を意味しているが、コンピュータに関するアーキテクチャは、正確な定義がなく、コンピュータやコンピュータ・システムに関する総称と考えればよいようである[68]。もともとは、機械語の設計のことであったとも言われる。アーキテクチャが使用されている用語として、ハードウェア・アーキテクチャ、IBM S/360 アーキテクチャ、ネットワーク・アーキテクチャ、アプリケーション・アーキテクチャ、そしてソフトウェア・アーキテクチャ等々がある。

3) アーキテクチャ・スタイル(Architectural Style)

システムの構成要素の論理的な配置と構成要素間の処理方式。アーキテクチャ・パターン(Architectural Pattern)とも呼ばれている[49]。

4) 処理構造(Processing Structure)

システムの構成要素の論理的な配置。

5) 処理方式(Processing Type)

システムの構成要素間の処理形態。

6) ソフトウェア・アーキテクチャ(Software Architecture)

ソフトウェア・システムがどうあるべきかを明確にするための、システム構築方針、必要なソフトウェア製品、機能群、インタフェース等を規定したもの。

1990 年代に各社より製品体系を示すために発表されるようになった。代表的なのが IBM 社の Open Blueprint である。最近では、ソフトウェア全体の構造をソフトウェア・アーキテクチャと広義に使用している[49]。

7) フレームワーク(Framework)

アーキテクチャほど厳密でないソフトウェア体系的な構成方法。

オブジェクト指向で使用されているフレームワークは、「抽象クラスの集合とそのインスタンス間の相互作用によって表現された、システムの全体または一部の再利用可能な設計」である[26]。本論文では、このオブジェクト指向で使用されている意味では使用しない。

8) プロプラエタリ(Proprietary)

各社の固有技術や製品。

9) ネットワーク・アーキテクチャ(Network Architecture)

データ通信システムに必要なサービス、すなわちセッションの設定・開放やデータの正

しい送受信など、を機能区分し、そのプロトコルを規定したもの。

1970 年代の IBM 社の SNA(Systems Network Architecture)がその代表である。

10) マーケティング(Marketing)

商品の生産から販売、サービスに至る一切の企業活動の総称[3]。

2. 論理型言語によるアプリケーション・モデリング

2.1 あらまし

ソフトウェア工学が提唱されてから数多くのアプリケーションのモデリング手法（設計手法）が提唱されている。現在のソフトウェア開発現場で採用されているモデリング手法の多くは、1980 年代の前半に活発な議論がなされている。その集大成的なシンポジウムが、昭和 59 年 4 月 10 日から 12 日にかけて「プログラム設計技法の実用化と発展」と題して情報処理学会主催で開催された。そのシンポジウムの報告として、まず情報処理学会誌の昭和 59 年 9 月号に、複合設計[34]、構造化プログラミングのワーニエ・メソド[83]と JSD[66]が掲載され、昭和 59 年 11 月号に、標準構造モデル[27]、SP-FLOW[97]、PAD/PAM[12]、Mother System[63]、DCG(Definite Clause Grammar)[39]と HISP[67]が掲載された。そして、シンポジウムでは議論されなかったが、新しいプログラム技法を使用して、シンポジウムで使用された共通問題[99]を解いた報告が情報処理学会誌昭和 60 年 5 月号に掲載された。掲載された技法又は言語名は、並列オブジェクト指向言語の ABCL[80]、論理型並列プログラミング言語の CP(Concurrent Prolog)[65]、属性文法をベースにした AG[28]、モジュラ・プログラミング言語の IOTA[103]、ストリームを扱う言語の Stella[35]、と関数型言語 Valid[1]である。

筆者らは、仕様記述法の味見と称して、仕様記述法の評価の一貫としてボーリング・ゲームの得点計算の仕様を CP や DCG を含む 10 種類の仕様記述言語で記述して、その使い勝手を報告している[29]。

情報処理学会誌で報告された方法論について、玉井によってモデリング手法がアーキテクチャ設計決定にどのように影響を与えているかの比較研究がなされている[86]。玉井に最も強い印象を与えた方法論は、CP と DCG であると報告している。この DCG によるモデリング手法は、Prolog 処理系の実行効率の問題や第 5 世代プロジェクトの終了と Prolog への関心の低下により、ビジネスの視点では全く進展が見られない方法論の一つになっている。しかし、21 世紀に入った現在、計算機処理能力の飛躍的な向上や Java に代表される解釈実行型言語に対する効率問題の技術改良により、Prolog 処理系の効率問題の垣根は低くなっている。また、Prolog による実用システムの稼働も報告されている。さらに、システム間で送受信されるデータの表現が XML 準拠に向かっている現在、ビジネスの視点でも情報技術の視点でも、1980 年代に提案した DCG によるアプリケーション・モデリング手法が XML と組合わさることによって新たな展開が開けると予想している。

本章では、酒倉庫問題を使用して、DCG によるアプリケーション・モデリング方法を概

説する[39]。DCG による厳密にプログラムを導出方法は、筆者らが電文解析問題を使用して報告している[38,89-91]が、事務処理問題へのモデリングの説明として、酒倉庫問題の方がより実世界の問題に近い例題である。まず、方法論の理解を助けるために、Prolog における DCG を説明する。次に共通問題と DCG の導出問題を説明する。そして今後の課題を論じる。

2.2 DCG

DCG は、文脈自由文法の自然な拡張である。DCG はプログラミング言語 Prolog の実行可能プログラムであるので、言語の記述だけでなく言語で記述された文字列を解析するためにも使うことが出来る。標準的な Prolog コンパイラを使って、DCG を Prolog にコンパイルすることが出来る。このことは DCG で実用的な言語アナライザーを実現できることを意味している[72,73]。

DCG は、[73]の中で次のように定義されている。DCG の文法規則の一般形は次の形式である。

```
LHS --> RHS.
```

これは、「LHS の可能な形式は、RHS である」を意味している。RHS と LHS とも、連結演算子','によってリンクされた複数の項目列である。

DCG は、次の方法で文脈自由文法を拡張する。

(1) 文法規則の左側は、非終端記号で構成され終端記号の列を後続することもできる。
(2) 文法規則の右側は、非終端か終端記号の列で構成する。
(3) 非終端記号は、変数や整数以外の Prolog 項である。
(4) 終端記号は、Prolog 項である。非終端記号と終端記号を区別するために終端記号の列は、Prolog リストとして文法規則の中に書かれる。終端記号が ASCII 文字コードならば、その様なリストは文字列として書くことが出来る。空列は、空リスト、[]または "" と記述する。
(5) Prolog 手続き呼び出し形式の例外条件は、文法規則の右側に含めることが出来る。その様な手続き呼び出しは、{}の中に書く。
(6) 選択肢は、分離演算子 ";" を使って、文法規則の右側に明示的に記述できる。

2.3 共通問題：酒倉庫問題

山崎が提案した酒倉庫問題は、次の通りである[99]。

『ある酒類販売会杜の倉庫では、毎日数個のコンテナが搬入されてくる。その内容はビン詰めの酒で、1つのコンテナには10銘柄まで混載できる。扱い銘柄は約200種類ある。倉庫係は、コンテナを受取りそのまま倉庫に保管し積荷票を受付係へ手渡す。また受付係からの出庫指示によって内蔵品を出庫することになっている。内蔵品は別のコンテナに詰め替えたり、別の場所に保管することはない。空になったコンテナはすぐに搬出される。

積荷票：コンテナ番号（5桁）
　　搬入年月、日時
　　内蔵品名、数量（の繰り返し）

さて受付係は毎日数10件の出庫依頼を受け、その都度倉庫係へ出庫指示書を出すことになっている。出庫依頼は出庫依頼票または電話によるものとし、1件の依頼では、1銘柄のみに限られている。在庫が無いか数量が不足の場合には、その旨依頼者に電話連絡し同時に在庫不足リストに記入する。また空になる予定のコンテナを倉庫係に知らせることになっている。倉庫内のコンテナ数はできる限り最小にしたいと考えているからである。

出庫依頼：　品目、数量
　　送り先名

受付係の仕事（在庫なし連絡、出庫指示書作成および在庫不足リスト作成）のための計算機プログラムを作成せよ。

出庫指示書：注文番号
　　送り先名
　　コンテナ番号
　　品名、数量
　　空コンテナ搬出マーク
　　（の繰り返し）

在庫不足リスト：　送り先名
　　品名、数量

・なお移送や倉庫保管中に酒類の損失は生じない。
・この課題は現実的でない部分もあるので、入力データのエラー処理などは簡略に扱ってよい。
・以上あいまいな点は、適当に解釈してください。』

方法論を理解しやすくするため、図 2-1 のような簡略化した共通問題を考える。

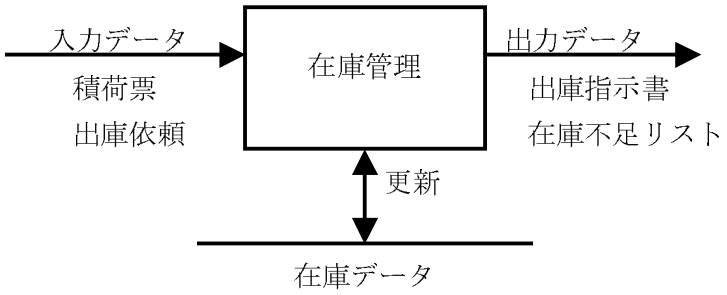


図 2-1　酒倉庫在庫問題

入力データは、積荷票（コンテナ番号、酒名、数量）と出庫依頼（注文番号、注文主、酒名、注文数量）である。出力データは出庫依頼に対する出庫指示書（注文番号、注文主、（コンテナ番号、酒名、数量、空コンテナ搬出マーク）の繰り返し）または在庫不足時の在庫不足リスト（注文番号、注文主、酒名、注文数量）である。積荷票が入力されるとコンテナと酒単位の在庫データを更新する。一件一酒銘柄の出庫依頼が入力されると在庫データを調べ、在庫がないか在庫量が注文数量を満たさないときは在庫不足リストを出力する。在庫量が注文数量を満たすと、出庫指示書を出力する。出庫指示書ではコンテナごとに出庫指示がなされ、空になる予定のコンテナであれば空コンテナ搬出マークを合わせて出力する。入力データの妥当性検査やコンテナを早く送りだすための工夫などは行わない。

2.4　DCG の導出

次のような手段を採用する。まず課題の入出力データの構文部分を BNF(Backus-Naur Form)によって記述し、これに入出力データ間に成立すべき関係などの意味部分を付加したものを DCG によって記述する。これを仕様と考える。DCG は、機械的に Prolog に変換できるので、変換された仕様は実行可能なものになる。すなわち、変換された Prolog プログラムが、Prolog の計算機構によって下降型構文解析／文生成プログラムとして動作するので、入力データの解析や出力データの合成が行えるのである。

2.4.1 入力データの構文記述

まず入出力データの具体構文を BNF によって記述する。図 2-2 および図 2-3 がそれである。

出力データについての構文記述は、出力結果を見やすくするために空白の挿入や行換えの指示等の非本質部分が混在して、多少読みにくくなっている。入力データの注文票の燥

返しに対応して出力構文<output>が繰り返され、出庫依頼が一件入力されればそれに対応して一件の<output>の処理ができるから、図 2-3 の出力データ構文において [..] で囲まれた構文は出力側 DCG に変換する時省略する。従って出力側 DCG は、出庫指示書及び在庫不足リストの構文のみが変換される。このように出力データ構文を縮小することによって意味記述のために導入する属性の数を減少させ、より自然な解にすることができる。

```
〈input〉      ::= 〈trans〉
〈trans〉      ::= 〈tran〉 〈trans〉 |
                  〈empty〉
〈tran〉       ::= 〈container〉 |
                  〈order〉
〈empty〉      ::= ''
〈container〉  ::= 'CONT' 〈c_no〉 〈sake〉 〈qty〉
〈c_no〉       ::= 〈integer〉
〈sake〉       ::= 〈identifier〉
〈qty〉        ::= 〈integer〉
〈order〉      ::= 'ORDER' 〈o_no〉 〈orderer〉 〈sake〉 〈qty〉
〈o_no〉       ::= 〈integer〉
〈orderer〉    ::= 〈identifier〉
```

図 2-2 入力データ構文

```
[   〈output_data〉 ::= 〈outputs〉                    ]
[   〈outputs〉     ::= 〈output〉 〈outputs〉 |        ]
[                      〈empty〉                      ]
    〈output〉      ::= 〈shipping_instruction〉 |
                       〈out_of_stock_notice〉
    〈shipping_instruction〉 ::= 'SHIP ' 〈s_o_no〉 〈s_orderer〉 〈n_l〉
                                〈s_items〉
    〈s_o_no〉      ::= 〈integer〉
    〈s_orderer〉   ::= ' ' 〈integer〉
    〈s_items〉     ::= 〈s_item〉 〈s_items〉 |
                       〈empty〉
    〈s_item〉      ::= 〈s_c_no〉 〈s_sake〉 〈s_qty〉 〈empty_mark〉 〈n_l〉
    〈s_c_no〉      ::= ' Container ' 〈integer〉
    〈s_sake〉      ::= ' ' 〈identifier〉
    〈s_qty〉       ::= ' ' 〈integer〉
    〈empty_mark〉  ::= '(Empty)' |
                       ' '
    〈out_of_stock_notice〉 ::= 'OUT_OF_STOCK'
                               〈unalloc_o_no〉
                               〈unalloc_oderer〉
                               〈unalloc_sake〉
                               〈unalloc_qty〉 〈n_l〉
    〈unalloc_o_no〉    ::= 〈integer〉
    〈unalloc_orderer〉 ::= ' ' 〈identifier〉
    〈unalloc_sake〉    ::= ' ' 〈identifier〉
    〈unalloc_qty〉     ::= ' ' 〈integer〉
    〈n_l〉             ::= 'CR/LF'
```

図 2-3 出力データ構文

2.4.2 意味の記述

1) 補助関数（述語）の準備

属性文法のやり方にならって、在庫状況を示す状態空間を考え、これを媒介して意味記述を行うことにする。この状態空間 St としてはコンテナ番号全体 Cont と酒名全体 Sake との直交空間 Cont×Sake を定義域とし、酒数量全体 Qty を値域とする関数 Cont×Sake →Qty ($(c, s)\rightarrow q$: コンテナ c に収納されている酒名 s の数量 q)の集合を考えればよい。この関数が入力データによってどう更新されるかを考えることになる。この状態空間 St は、Prolog を用いて次のように実現する。

St を 3 つ組 stock(c,s,q)の集合とする。St の初期状態は空集合 ϕ とする。集荷票の入力に際して、それから定まる c,s,q により 3 つ組 stock(c,s,q)を定め、St∪{stock(c,s,q)}→St を行う。出庫依頼の入力に際しては、それから誘導される order(s,q) によって St を更新する。またある Co∈Cont に対して St(Co) = { stock(c,s,q)∈St | C = Co }で、St(Co)の q たちの総和が 0 になれば St − St(Co) → St を行う。

以上の St への追加、変更、削除をそれぞれ add_stock(C,S,Q)、change_stock(C,S,Q,NQ) および delete_stock(C)の述語によって行うようにする。

付随する補助述語としてつぎを用意する。

sake_stock(S,T)： 倉庫内にある酒名 S の総量 T

is_empty_cont(C)： コンテナ C は空

以上の Prolog プログラムは図 2-4 のとおりである。ここで、

bagof(X,P,S)： S は P を満足するすべての X のつくる多重集合である。

Y^Q： Q を満足する Y が存在する。

assert(C)： 集合（データベース）に C を追加する。

retract(C)： 集合から C を削除する。

という Prolog 組込みの述語を使用する。

```
add_stock( C,S,Q )    :- assert( stock( C,S,Q ) ).
change_stock( C,S,Q,NQ ) :- retract( stock( C,S,Q ) ),
                            assert( stock( C,S,NQ ) ).
delete_stock( C )   :- retract( stock( C,_,_ ) ), fail.
delete_stock( _ ).
sake_stock( S, T ) :- bagof( Q, C^stock( C,S,Q ), L ),
                      total( L, T ).
sake_stock( S, T ) :- T is 0.
is_empty_cont( C ) :- bagof( Q, S^stock( C,S,Q ), L ),
                      total( L, T ), !, T = 0.
total( [], T )      :- T is 0.
total( [X|Y], T )   :- total( Y, New_T ), T is X + New_T.
total( X, T )       :- T is X.
```

図 2-4 補助述語

2) 意味の記述

BNF を用いた構文記述をもとに、これに入出力データ間に成立すべき関係などを付けて意味記述を完成させる。このとき DCG を用いて書けば、図 2-5 および図 2-6 のようなプログラム部分が得られる。意味を記述するために構文記述に付けられた属性を説明する。

C: コンテナ番号
S: 酒名
Q: 数量または注文数量
N: 注文番号
O: 注文主
T: 酒名 S の総量
U: 出庫指示票作成のための注文数量残
NU: 同上（更新用）
OQ: 注文累の注文数量
SQ: 出庫指示票内の数量
NQ: 在庫データ更新のための数量

以上を、Prolog プログラムとして完成させるために、若干の段取り（house-keeping）部分（図 2-7）を補足追加する。段取り部分では、

see(*F*): 入力ファイル *F* を開く。
seen: 入力ファイル *F* を閉じる。
get(*K*): 現在の入力ファイルより空白でない一文字を *K* に読み込む。
get0(*K*): 現在の入力ファイルより一文字を *K* に読み込む。
¥+*P*: not *P*

という Prolog 組込み述語を使用する。

以上の仕様は Prolog プログラムに変換することによってプログラムとみなすことができる。

```
input --> trans.
trans --> tran, !, trans.
trans --> empty.
tran  --> container |
          order.
empty --> "".
container --> ['CONT'], c_no( C ), sake( S ), qty( Q ),
              { add_stock( C, S, Q ) }.
c_no( C ) --> [C].
sake( S ) --> [S].
qty( Q ) --> [Q].
order --> ['ORDER'], o_no( N ), orderer( O ), sake( S ), qty( Q ),
          { make_output( N, O, S, Q ) }.
o_no( N ) --> [N].
oderer( O ) --> [O].
```

図 2-5 入力側 DCG

```
output( N,O,S,Q ) --> { sake_stock( S, T ) },
       ( ( { T >= Q }, shipping_instruction( N,O,S,Q ) ) |
         ( { T <  Q }, out_of_stock_notice( N,O,S,Q ) )   ).
shipping_instruction( N,O,S,Q ) -->
       { write( 'SRIP ' ) }, s_o_no( N ), s_orderer( O ), n_l,
       s_items( S,Q,U ).
s_o_no( N ) --> { write( N ) }.
s_orderer( O ) --> { write( ' ' ), write ( O ) }.
s_items( S,Q,U ) --> { Q ¥== 0 }, s_item( S,Q,U ), !,
                                  s_items( S,U,NU ).
s_items( _,_,_ ) --> [].
s_item( S,OQ,U ) -->
     { stock( C,S,Q ), Q > 0,
       ( ( Q  > OQ, SQ is OQ, U is 0,       NQ is Q - OQ ) |
         ( Q =< OQ. SQ is Q,  U is OQ - Q, NQ is 0      )  ),
       change_stock( C,S,Q,NQ ) },
     s_c_no( C ), s_sake( S ), s_qty( SQ ), empty_mark( C ), n_l.
s_c_no( C ) --> { write( ' Container ' ), write( C ) }.
s_sake( S ) --> { write( ' ' ), write( S ) }.
s_qty( SQ ) --> { write( ' ' ), write( SQ ) }.
empty_mark( C ) --> { is_empty_cont( C ), delete_stock( C ),
                      write( '(Empty)' ) }.
empty_mark( _ ) --> { write( ' ' ) }.
out_of_stock_notice( N,O,S,Q ) --> { write( 'OUT_OF_STOCK ' ) },
                                   unalloc_o_no( N ),
                                   unalloc_orderer( O ),
                                   unalloc_sake( S ),
                                   unalloc_qty( Q ), n_1.
unalloc_o_no( N )    --> { write( N ) }.
unalloc_orderer( O ) --> { write( ' ' ), write( O ) }.
unalloc_sake( S )    --> { write( ' ' ), write( S ) }.
unalloc_qtY( Q )     --> { write( ' ' ), write( Q ) }.
n_1 --> { nl }.
```

図 2-6 出力側 DCG

```
stock_control ( Fname ) :- get_file( Fname, In ),
                           input( In, [] ).
make_output ( N, O, S, Q ) :- output( N, O, S, Q, Temp, [] ).
get_file( F, L ) :- see( F ), tokens( L ), seen.
tokens( [T|N] ) :- token( T ), !, tokens( N ).
tokens( [] ).
token( T ) :- get( K ),
              ( ( isdigit( K ), digits(U)        ) |
                ( isletter( K ), alphanums( U )  ) ),
              name( T, [K|U] ).
isdigit( X ) :- ¥+( atom( X ) ), X > 47, X < 58.
isletter( X ) :- ¥+( atom( X ) ), ( ( X > 96, X < 123 ) |
                                    ( X > 64, X < 91  ) ).
digits( [K|U] ) :- get0( K ), isdigit( K ), !, digits( U ).
digits( [] ).
alphanums( [K|U] ) :- get0( K ), alphanum( K ), !,
                      alphanumes( U ).
alphanums( [] ).
alphanum( K ) :- isletter( K ) |
                 isdigit( K ).
```

図 2-7 ハウスキーピング・プログラム

2.5 まとめと今後の課題

本章で提案したモデリング手法は、一般的に文章処理や事務処理の問題に応用できる。問題の入出力データの要求仕様を DCG で表現する。入力データや出力データに対する要求仕様や入出力データ間に成立する関係の要求仕様を Prolog で表現する。DCG を組み入れた Prolog を利用して簡単にプログラムが作成でき、そのまま実行することが出来る。構造化手法やオブジェクト指向等のアプリケーション・モデリングの手法と異なり、このモデリング手法は、問題の要求仕様をほぼ機械的にプログラムに変換できるので Rapid-prototyping 手法として有効である。

21 世紀になり、システム間で受け渡されるデータ形式として、XML が注目を集めている。XML で表現されたデータ表現を DCG に変換することによって同じモデリング手法が適用できると予想している。

情報技術の進展により、Prolog 処理系も実行効率の問題が大きな障害にならなくなっている。実際、Prolog で開発された実用システムが報告されている。従って、XML と DCG を使用して、ある程度の規模の事務処理システムを構築評価することが今後の課題である。

3. ソフトウェア・アーキテクチャの発展

3.1 あらまし

本章では、ソフトウェア・アーキテクチャの発展の経緯を技術的側面とビジネス的側面を中心に、その出現の背景、経緯、そして現状を概観し、そして今後の方向性を論じる。まず、1980 年代の後半に出現した IBM 社の S/360 に代表されるメインフレーム時代のアプリケーション・アーキテクチャを概観し、1990 年始め頃から始まったオープンを標語にした UA(Unisys Architecture)等の新しいソフトウェア・アーキテクチャを論じる。そして、1990 年代中頃の本格的なオープン時代の各社のフレームワークの動向を解説し、最後に 21 世紀のソフトウェア・アーキテクチャのあるべき姿を論じる。

3.2 囲い込み時代

1980 年代までのコンピュータ業界は、IBM 社が新しい概念を発表し、BUNCH（バロース、ユニバック、NCR、CDC、ハニウェル）と総称されたメーカおよび国産メーカが対応した概念を少し遅れて発表するといったことの繰り返しであった。例えば、IBM 社が 1974 年 9 月にネットワーク・アーキテクチャである SNA(Systems Network Architecture)を発表すると、スペリー・ユニバック社（現、ユニシス社）は 1976 年 11 月に DCA(Distributed Communications Architecture)を、バロース社（現、ユニシス社）は 1979 年に BNA(Burroughs Network Architecture)を追いかけて発表している。

1987 年当時、IBM 社は当時 11 種類の異なるハードウェアで 14 種類の異なる OS をもっていた。同社は、自社の異なるハードウェア・アーキテクチャに基づくコンピュータ間で、アプリケーション・プログラムの移植性（可搬性）、接続性、操作の一貫性を得るために、SAA (Systems Application Architecture)を発表した。SAA は、アプリケーション・プログラム回りのインタフェースである共通プログラミング・インタフェース、共通コミュニケーション・サポート、及び共通ユーザ・アクセスを規定している。図 3-1 にその概念図を示す。ユーザを自社のアプリケーション・アーキテクチャ内に封じ込めて売り上げの拡大をはかるのが真の目的であったと思われるが、ユーザにアーキテクチャという投網をかける IBM 社の遠大なマーケティング戦略であった。

国産各社を含め各計算機メーカは、同じ様な概念の対応するアプリケーション・アーキテクチャを矢継ぎ早に市場に投入してきた。表 3-1 に各社の対応とそのアーキテクチャ名を示す[59,60,61]。

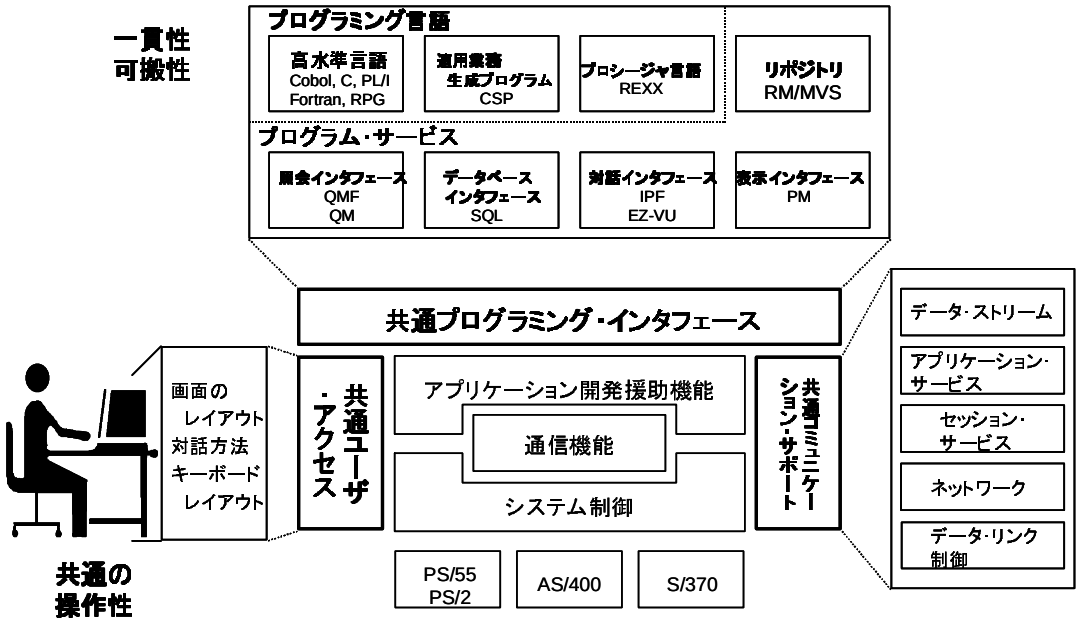


図 3-1　SAA

表 3-1　1980 年代の各社のアプリケーション・アーキテクチャ

発表年月	会社名	アーキテクチャ名
1987.4	IBM 社	SAA(Systems Application Architecture)
1987.5	富士通	SIA(Systems Integration Architecture)
1987.7	日立製作所	HAA(Hitachi Application Architecture)
1987.11	日本電気	DISA(Distributed Information processing Systems application Architecture)
1988.1	DEC 社	AIA(Application Integration Architecture)
1988.10	沖電気	DAA(Distributed Application Architecture)

現在、これら各社のアプリケーション・アーキテクチャは、オープン時代の夜明けとともにメインフレーム中心という概念ゆえにその市場性と話題性を失い、歴史上の用語として登場するのみになっている。

SAA の発表に刺激されて国際標準化機構(ISO)の場でも、アプリケーション・プログラムの移植性に目を向けたアーキテクチャ作りが 1988 年 12 月に日本が主導権を持った形で作業が開始され、1991 年 5 月に IAP(Interface for Application Portability)フレームワークとして最終報告がなされた[20]。この報告書で、重要な概念であるアプリケーション・ソフトウェアとアプリケーション・ソフトウェア・プラットフォームおよび外部世界、そしてそ

れらの間の API(Application Programming Interface)と PEI(Platform External Interface)の概念が導入され、標準化すべき機能の切り分けがなされた。図 3-2 にその外観を示す。

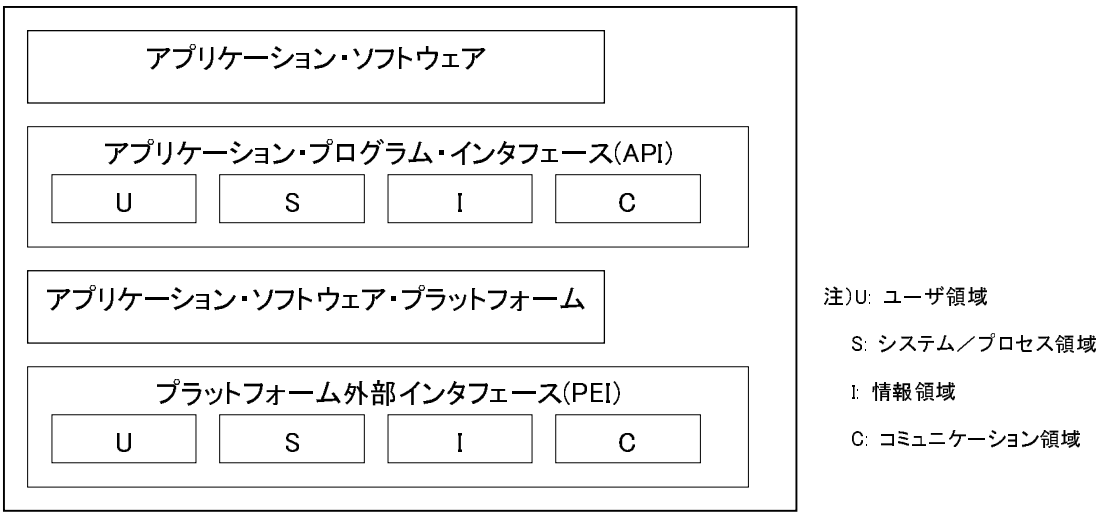


図 3-2　IAP フレームワーク

ほぼ同じ時期に、米国電気電子学会(IEEE)に於いてオペレーティング・システム(POSIX)の標準化作業が進行しており、IAP フレームワークはその中の OSE(Open System Environment)の参照モデルに大きな影響を与えている。OSE 参照モデルは 1994 年 5 月にドラフト 16.1 版が ISO に技術報告書の標準候補として提案され、1996 年 12 月に ISO に制定された[23]。この参照モデルの機能区分がソフトウェア・アーキテクチャ、例えばオープン・グループの TOGAF(The Open Group Architectural Framework)等、に影響を与えている[70]。図 3-3 に OSE 参照モデルを示す。IAP フレームワークの四つの API 領域であるユーザ領域、システム／プロセッサ領域、情報領域及びコミュニケーション領域は、それぞれ OSE 参照モデルの四つの API 領域であるヒューマン／コンピュータ対話サービス、システム・サービス、情報サービス及びコミュニケーション・サービスに対応している。

標準化の動きとは別に、各社が SAA に対抗するアーキテクチャを発表していた 1980 年代末は、ユニシス社は、丁度スペリー社とバロース社の合併の時期で対応するアーキテクチャを発表する状態ではなかった。すなわちスペリー系のアーキテクチャとバロース系のアーキテクチャを統合するアプリケーション・アーキテクチャを市場に出せない状態であった。結果的には、オープン時代の到来で話題性がなくなる SAA 擬きのアプリケーション・アーキテクチャを出せなかったことは、ビジネス的に見て不幸中の幸いであった。

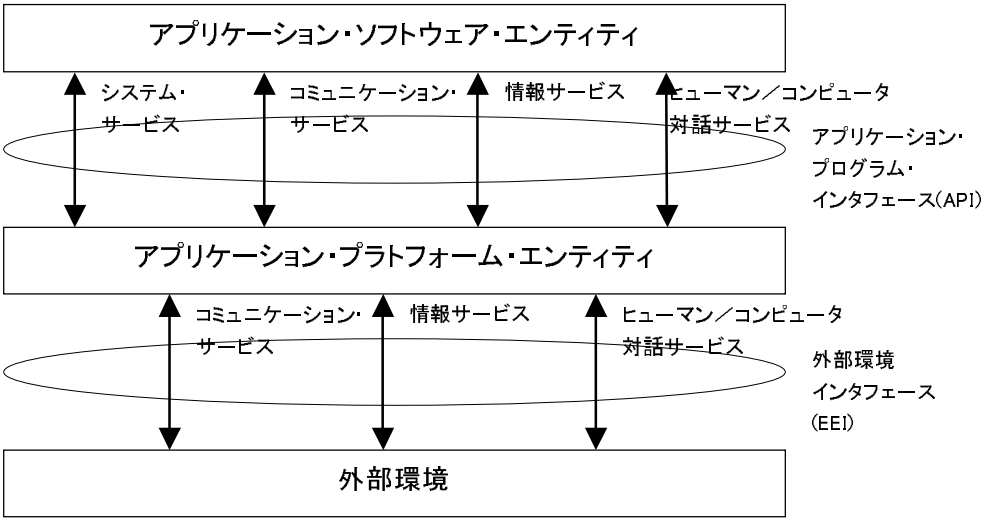


図 3-3　OSE 参照モデル

3.3 オープン時代の夜明け

1990 年 2 月に NCR 社は、それまでのメインフレーム路線を捨て、Unix に傾注した新しいソフトウェア・アーキテクチャ、OCCA(Open, Cooperative Computing Architecture)を発表した[10]。「オープン」を前面に出した新しいアーキテクチャの出現である。表 3-2 に 1990 年代前半に各社が発表したソフトウェア・アーキテクチャ等の一覧表を示す。

表 3-2　1990 年代前半の各社のソフトウェア・アーキテクチャ

発表年月	会社名	アーキテクチャ名
1990.2	NCR 社	OCCA(Open Coperative Computing Architecture)
1990.11	ユニシス社	UA(Unisys Architecture)
1991.1	NTT	MIA(Multivendor Integration Architecture)
1991.5	ISO	IAP(Interface for Application Portability)
1991.11	IEEE	Posix OSE(Open Software Environment)
1992.6	日本 IBM	Open Vision
1993.10	NTT 他	SPIRIT(Service Provider's Integrated Requirements for Information Technology)
1994.3	富士通	OPEN*framework*
1994.4	IBM 社	Open Blueprint

ソフトウェア・アーキテクチャは、アプリケーション・プログラムの移植性や接続性、操作の一貫性を得るためのアプリケーション・アーキテクチャを包含して、ネットワークシステムがどうあるべきかを明確にするための、システム構築方針、必要なソフトウェア

製品、機能群、インタフェースの提示等を示したものである。図 3-4 にソフトウェア・アーキテクチャとアプリケーション・アーキテクチャの関係を示す。

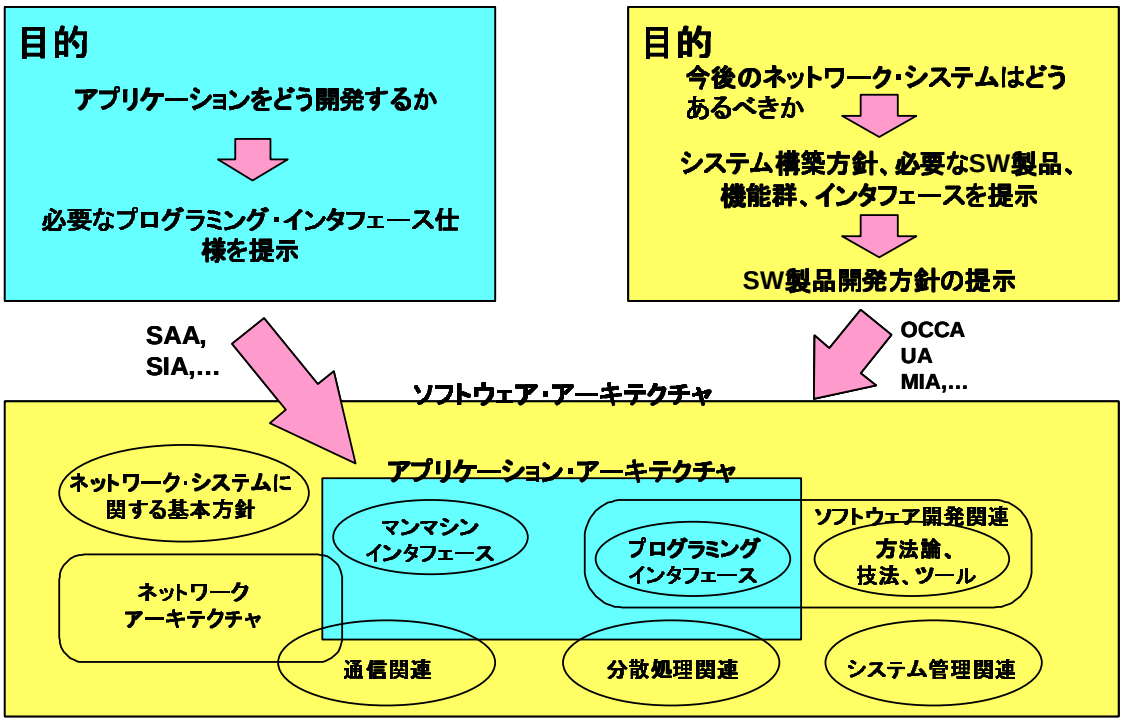


図 3-4　アプリケーション・アーキテクチャとソフトウェア・アーキテクチャ

1989 年末、ユニシス社では、スペリー系とバロース系のアーキテクチャを統合する新しいソフトウェア・アーキテクチャ作成に二つの動きが潜行していた。一つはワークステーション(ユニシス社の DW2 シリーズ)よりの統一した開発環境である SGE(Solution Generation Environment)であり、もう一つは DCA のアーキテクトであった H.Foxwell 氏を主アーキテクトとした ECA(Enterprise Computing Architecture)であった。ECA は発表時に UA と名称変更された。前者は、2 系統のアーキテクチャを統合するアーキテクチャとしてはカバーする機能範囲が狭く、オープン時代のアーキテクチャになる可能性が低いと判断された。

1990 年 4 月 4 日から 6 日の間、米国フィラデルフィアのユニシス本社近くのホテルに筆者を含め 40 名近い人が全世界より集まり、第 1 回の UA ワークショップが開催され、発表に向けての作業が開始された。ユニシス社が採用した戦略は、今までのユニシス・メインフレーム上の資産と共存しながら「オープン化」する戦略であった。ユニシス社が 1990 年 10 月に、少し遅れて日本ユニシスが 1990 年 11 月に UA を発表した。

UA がイメージしたのは、1990 年代中頃の複数の異機種コンピュータが共存しながら、ユーザが簡単に情報資源を利用できる統合的なネットワーク環境であった。狙いとしては、異機種間の相互運用性の向上、ユーザ・インタフェースの共通化、開発生産性の向上、システム管理の統合化、標準化が未成熟な又これから発展する分野（例えば、CASE、4GL、高速トランザクション処理等）の業界最先端の高付加価値サービス、ユーザ資産の継承、等であった。UA では、プラットフォームを役割によって、インフォメーション・ハブ、ネットワーク・サーバ／地域サーバ、ワークステーション／ワークグループ、の３種類に分類し、それぞれが備えなければならない機能の属性を明確にし、メインフレームのイメージの払拭をはかった。また、オープン・クラス、プレミアム・クラス、共存クラスと機能を分類し、標準ベースの機能とユニシス固有の機能と他社との共存用機能を明確にした。そして、分散処理システムの構築に必要な様々な機能を図 3-5 で示すような五つのサービスに体系化した[53,95]。各サービスはさらにサブサービスと、そしてさらに詳細に分類され、各サービスの機能および準拠する標準規格等が明確にされた[54-58]。

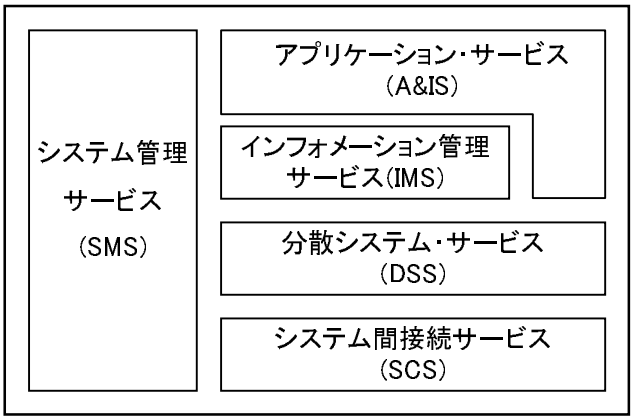


図 3-5　UA の五つのサービス

UA は 1990 年代中頃のオープンな分散処理システムを前面に打ち出したソフトウェア・アーキテクチャであったが、Unisys Architecture のユニシスという名称と３種類のプラットフォームの導入によりプロプラエタリなメインフレーム中心のアーキテクチャであるというイメージを払拭できず、またアーキテクチャに準拠したオープンな製品の提供、4GL 等の有力なソフトウェアのオープン化の遅れ、そしてアーキテクチャそのものの活用方法の提示が時期を得ず、次第に市場での影響力や話題性を失っていった。ユニシス社の企業力の衰退とも大きく関連するが、その後のオープン市場の動きや他社の動きから判断すると、ビジネス的に見て UA の市場への投入は３～４年早過ぎたと思われる。ユニシス社は、情報技術の進展に合わせて、UA を 1995 年に OTF(Open Technology Framework)[96]と名

称を変更し、ミドルウェアのアーキテクチャへとその位置付けを変更した。お客様の情報システムのインフラストラクチャの構築を支援するアーキテクチャ・サービスの一貫としてのミドルウェアの技術体系と位置付けて利用している。

1990 年代に入り、オープン化の波が本格化し、IBM 社も対応を迫られるようになった。1990 年 9 月に SAA の世界と Unix の世界とのインターオペラビリティ（相互運用性）を発表し、「IBM はオープンです」というオープン宣言を発表している。さらに、日本における「オープン」の高まりに対応するため、日本 IBM が独自にオープン・システム構築のために指針を体系的に集大成して「OPEN VISION」を 1992 年に発表した[14]。これらの発表は、1994 年 4 月に IBM 社が発表した Open Blueprint までの繋ぎのマーケティング戦略であった[51]。

Open Blueprint の概説書に書かれている事柄を見ると、Open Blueprint を UA と置き換えても何ら違和感のないものであり、メインフレーム・メーカがオープン化に変遷する時の戦略に同一性を見ることができる。IBM 社のホームページ等で公開されたドキュメントは、さすが IBM 社と思わせる内容になっている。その後のソフトウェア・アーキテクチャの構成方法に大きな影響を与えている。Open Blueprint では、分散アプリケーション環境を設計・構築・利用・管理するために必要なサービスを大きく、アプリケーションおよびアプリケーション・イネーブリング・サービス、分散システム・サービス、ネットワーク・サービス、およびシステム管理サービスに分類している。この分類方法は、その後ほぼ標準的なものとして各社のソフトウェア・アーキテクチャで使用されることになる。

Open Blueprint は毎年改訂されており、出版物の最新版は 1996 年 12 月の第三版である[17]。第三版では技術の進歩に追随して、ウエブ・ブラウザ、Java 仮想マシンや HTTP 等が取り込まれている。図 3-6 に Open Blueprint の第三版の概要図を示す。その後、Open Blueprint の Web 版が公開され、1997 年１０月には、第四版がホームページ上で公開されていたが、現在 IBM 社のホームページでは、Open Blueprint の資料を見ることが出来ず、その使命は終了している。

1995 年中頃、ソフトウェア・アーキテクチャなどの概念の形成は、外国コンピュータ・メーカの独壇場である。国産の雄である富士通は、オープン・システム環境でシステムの統合化を実現する方法論として 1994 年 3 月に OPEN*framework* を発表した。OPEN*framework* は富士通の子会社である英国 ICL 社が開発したもので、情報技術を八つの要素に分類している。図 3-7 に OPEN*framework* の八つの技術要素を示す。

これらのソフトウェア・アーキテクチャは、1990 年代後半のフレームワーク戦国時代において、生き残りのための追加修正をはかっているが次第に市場で話題にならなくなっている。これらとは別に米国の大学や研究所で新しい技術開発や方向転換がなされていた。

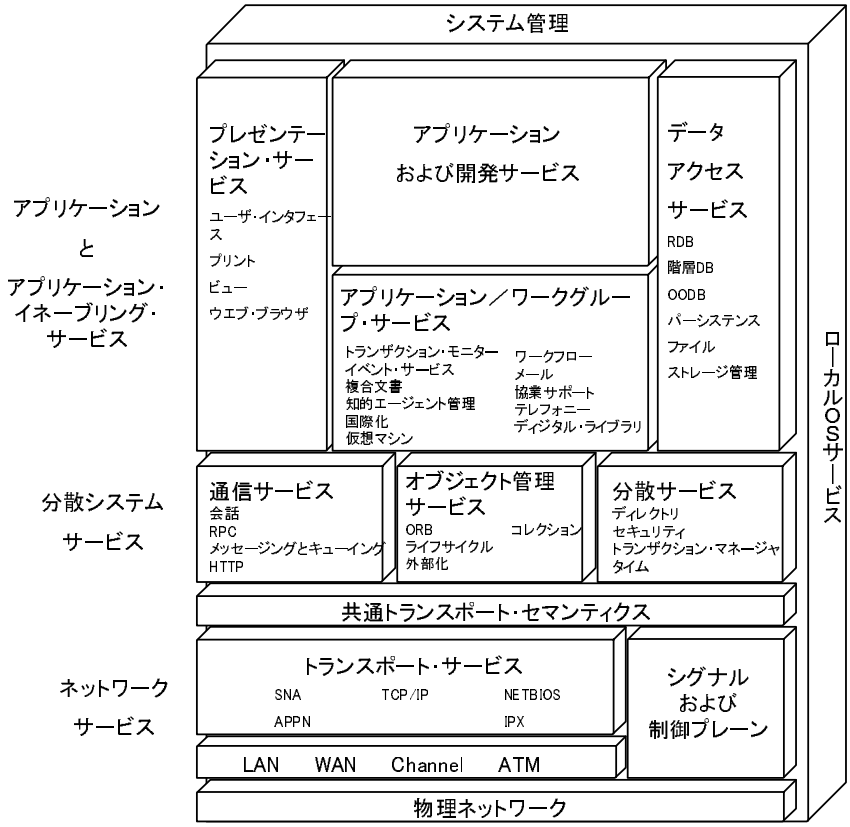


図 3-6　Open Blueprint

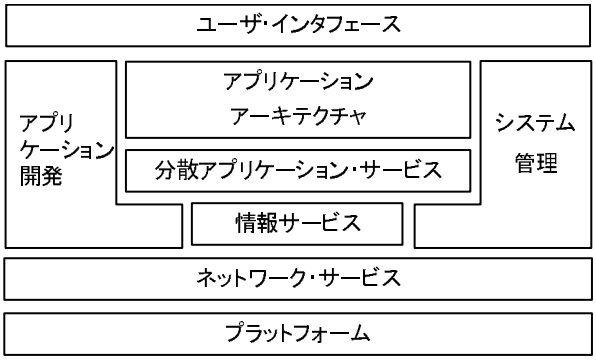


図 3-7　OPEN*framework*

新しい動きで、1990 年代後半に大きな影響を与えたのが、1991 年に実施されたインターネットの商用化と 1993 年イリノイ大学 NCSA のインターネットのウエブ・ブラウザである Mosaic の開発およびそれに続く Netscape 社の立ち上げであった。

3. ソフトウェア・アーキテクチャの発展

1990年代始めの米国でのオープン化の波が数年遅れて日本でも始まった。オープン化の波にもまれながら、早い段階から日本ユニシスはUnixをベースにオープン市場に於けるビジネスを展開していた。1995年頃になると、オープン市場の開発環境はクライアント／サーバ開発環境と同一視されるようになり、市場には数多くのクライアント／サーバ用のソフトウェアが市場に氾濫するようになった。ユーザがこれらの多くの市販のソフトウェアを活用してアプリケーションを構築する時、使用するソフトウェア間での相互接続性や相互運用性、いわゆるソフトウェア間の整合性検査に多くの時間と労力を費やすようになった。またお客様に提案書を提出するとき、コンサルテーションをするとき、アプリケーションの構築を支援するとき、ソフトウェア・プロダクト間の整合性の経験・知識が必須のものになってきた。どの様な業務に、どの様なプロダクト群が適合し、それらプロダクト間の整合性に問題がないのか。何故この組み合わせが最適なのか。その理由は、等々である。知識と経験の共有化、全体としてのサポート・コストを削減するため、何らかのクライアント／サーバ・システム構築のためのソフトウェア・アーキテクチャ、クライアント／サーバ・システムの処理形態のモデルと、それに基づくプロダクトの組み合わせを包含したフレームワークが必要になってきた。

1995年4月に、新たな組織が日本ユニシス社内に設けられ、クライアント／サーバ・システム構築のためのフレームワーク作りを開始した。目標は、クライアント／サーバ構築用のオープンなソフトウェア・プロダクトの組み合わせであるプロダクトセットの導入とそのプロダクトセットを選択するための考え方の作成であった。

作成された考え方は、オープン・ソリューション・フレームワーク(OSFW)と命名され、大きく四つの要素より構成された。それらは、企業情報システム・モデル、クライアント／サーバ・ソリューション(C/SS)アーキテクチャ（初版はC/SSフレームワーク）、C/SSモデル、そしてオープンソフトウェア・プロダクトセットである。OSFWの技術的背景としては、1990年10月のUA、1992年5月に発表した開発環境のフレームワークであるASDF(Advanced Solution Development Framework)、1993年3月に発表した分散処理システムの実行環境のフレームワークであるACCF(Advanced Cooperative Computing Framework)、1994年2月に発表したシステム管理のフレームワークであるASMF(Advanced System Management Framework)がそのベースになっている。設計目標として最も注意が払われたのは、フレームワークは、絵に描いた餅でなく、現在利用可能な情報技術とソフトウェア・プロダクトを使用して作ることであった。OSFWは、1995年10月に社内発表がなされ、明けて1996年1月に、ユニシス社のメインフレームの新モデルであるItascaと同期して社外発表がなされた[40,41]。

OSFWの設計思想及び各構成要素は、第4章で詳細に論じる。

3.4 フレームワーク戦国時代

ユニシス社が提言したUAは、1990年代中頃の分散処理システムのイメージを描いていた。そのイメージは、時代を先読みした的確なイメージであったといえる。しかし、分散処理システム構築のために予想していた技術は、ISO標準であり、通信はISOのOSIであり、GUIはMotifやウィンドウズであった。しかし、1990年代中頃に話題になっていたのは、TCP/IP上のHTTPプロトコルであり、インターネットであり、Webブラウザであり、Javaである。1990年には誰も予想しなかったものが、業界標準として使われていた。UAは、OTFのアーキテクチャへと変遷して、技術体系を最新の技術に適合するよう改訂して、生き残りを図ろうとした。IBM社のOpen Blueprintも同様な改訂がなされ、アーキテクチャとしての位置付けを保ち続けようとした。第4章で詳細するOSFWも例外ではない。各社のビジネス戦略と直結したソフトウェア・アーキテクチャは、市場の動向及び情報技術の動向に併せて、新しい技術、新しい製品を取り込まないかぎり、直ぐに時代遅れのものになってしまう。このことは、この種のマーキテクチャ（マーケティング・アーキテクチャ）の宿命である。

OSFWを発表した1995年以降、インターネットやWebやJava等が常識の世界に突入した。コンピュータの基盤技術であるソフトウェア・アーキテクチャにおけるコンピュータ企業間の競争は、新しい技術の出現で実現方法に若干の進歩は見られるものの、ネットワーク、分散処理、アプリケーション開発・実行、およびシステム管理とほぼ体系的に固まったものとなり、コンピュータ業界は新しい領域のビジネスを展開し始めた。それは、GUIの分野で成功を収めたオブジェクト指向技術の次なる展開であるアプリケーションの部品化（ビジネス・オブジェクト）である。21世紀には本格化すると思われるビジネス・アプリケーションの部品化を目指して、OMGによるビジネス・オブジェクトの標準化[69]を一つの中心として、各社が新しいソフトウェア・アーキテクチャを次々と発表し、それぞれのアーキテクチャは普及のために同盟関係を結成し、主導権争いを開始している。もう一つの中心は、マイクロソフト社のActiveXである。

表3-3に1990年代後半に発表された各社のフレームワークの一覧表を示す。

1996年8月にIBM社は、Javaベースのビジネス・オブジェクト用のフレームワーク、サンフランシスコ・フレームワーク(San Francisco Framework)、を発表した。パートナー企業との同盟関係を積極的に拡大している[7]。1997年8月には、第1段のビジネス・オブジェクト群の製品提供が同盟企業に対して開始されている。適用分野が拡大され、ビジネス・オブジェクトが追加されるに従って、徐々に業界に影響を与えていたが、最近の評価は、マーケティング的に失敗プロジェクトであったようである。

表 3-3　1990 年代後半に発表された各社のフレームワーク

発表年月	会社名	フレームワーク名
1996.1	日本ユニシス	オープン・ソリューション・フレームワーク(OSFW)
1996.7	Netscape 社	ONE(Open Network Environment)
1996.8	IBM 社	San Francisco Framework
1996.10	オラクル社	NCA(Network Computing Architecture)
1997.1	日本ユニシス	EC フレームワーク
1997.4	IBM 社	NCF(Network Computing Framework)
1997.9	マイクロソフト社	WindowsDNA
1997.10	日本電気	WCF(WebComputing Framework)
1997.11	富士通	SolutionVision
1998.1	東芝	C Solution
1998.3	サン社	EJB(Enterprise Java Beans)
1998.4	日立	Network Objectplaza
1998.8	日立	Cyberfront
1999.4	日本ユニシス	OnNet Solution
1999.7	富士通	Internet ビジネス・ソリューション
1999.11	日本電気	iBestSolutions

OS 周りのソフトウェア・アーキテクチャが整備されるに従って、ソフトウェア・アーキテクチャの議論の焦点は、ミドルウェアからアプリケーション領域へと拡大してきた。ハードウェア部門も独自の OS も持たないアプリケーション・ベンダーやデータベース・ベンダーは、自社のソフトウェアの業界内での生き残りと地位を確保するために、自社の戦略ソフトウェアを位置付けたソフトウェア・アーキテクチャを発表し始めた。

1996 年 7 月にネットスケープ社は、オープン・クライアント／サーバ・アプリケーション開発プラットフォームとして Netscape ONE(Open Network Environment)を発表し、同時に多くの開発ベンダーの賛同を得ている。ネットスケープ社は、ネットスケープ・ナビゲータをその基盤としようとしているが、マイクロソフト社の金に物を言わせたインターネット・エクスプローラの提供により、先の見えない戦いを強いられていたが、アメリカン・オンライン（AOL）社により買収され、かろうじて生き延びている。

1996 年 10 月にオラクル社は、彼らのデータベースを中心にした新しいアーキテクチャ、NCA(Network Computing Architecture)を発表した。クライアント上のソフトウェア、アプリケーション・サーバ上のソフトウェア、データベース上のデータをカートリッジという概念、いわゆるオブジェクト、で自由に Plug&Play できる環境を定義した。これは、ソフトウェアの部品化のためには、正しい方向に見えるアプローチであるが、オラクル社が何処まで彼らが提供する製品を速やかに NCA に変更できるかが鍵であり、どれだけ賛同者

が集められるかも重要である。図 3-8 に NCA の概念図を示す[71]。

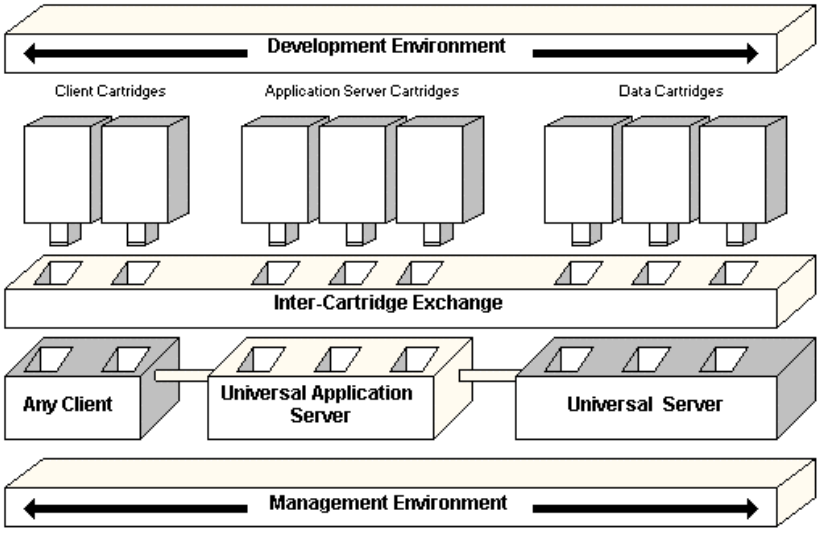


図 3-8　Network Computing Architecture

この年の新しい動きとしては、ERP の代表メーカである SAP 社がビジネス・フレームワークを発表し、アプリケーション部品の主導権を取ろうとしている[24]。アプリケーション部品ビジネスの萌芽である。

1997 年 4 月に IBM 社は、電子ビジネスのフレームワークとして NCF(Network Computing Framework)を発表した[64]。この頃より IBM 社は、1999 年中頃より大々的にイメージ戦略として打ち出す e-business を使用し始めている。大きく彼らのマーケティング戦略を電子ビジネス（電子商取引ビジネス、インターネット・ビジネス、e ビジネス等が同様な意味で使用されている）に焦点を当て始めた。

マイクロソフト社は、アクティブ・デスクトップとアクティブ・サーバ上の機能と標準プロトコルを定義した ActiveX のフレームワークそして次に COM+を発表し、OMG の CORBA 等を向こうにまわして､我こそは情報技術の世界標準であると我が道を歩んでいる。何年か前の IBM 社の姿である。デスクトップ(PC)のオペレーティング・システムを完全に押さえた現在、これが世界標準であるという事実は当面否定できない。オープン・ソースのソフトウェアである Linux の台頭で新しい次の業界標準獲得に向けての動きが始まっている。21 世紀には、異なる新しい企業が主役の座に座っているかもしれない。

ソフトウェア・アーキテクチャという視点から見るとアプリケーションを稼働させるための基盤部分、即ちオペレーティング・システム、ネットワーク、分散処理やデータベース等のミドルウェア、の体系化は、ほぼ必要な機能群が明確になり、実装するソフトウェ

アの入れ替え戦は今後とも発生するが大きな流れは決まった。ユーザに相対するデスクトップはマイクロソフトが制覇し、通信はOSIでなくTCP/IPが押さえ、Unixやメインフレームは OMG の CORBA 準拠のミドルウェアでマイクロソフトのアップサイジングに対抗することになる。現在のソフトウェア・アーキテクチャの戦場は、アプリケーション・プログラムのオープン化である。ビジネス・オブジェクトやアプリケーション部品の用語で代表される新しいビジネス領域に、コンピュータ・メーカやソフトウェア・ベンダー各社がユーザを巻き込んで参入している。既に述べたIBM社のサンフランシスコ・フレームワークの同盟関係や、多くの企業が参画するOMGのビジネス・オブジェクト活動や、CIM業界でのアプリケーション・フレームワークによるアプリケーション部品の定義など、様々な企業や団体が縦横に入り交じって活動している。どの企業がどの団体が勝ち馬になるのかは不明であるが、一社だけでは何もできないことは明白な事実である。

1997年10月に、日本電気がWCF(WebComputing Framework)を発表し[33]、同年11月には、富士通がSolutionVisionを発表した[102]。図3-9にSolutionVisionの全体像を示す。ソリューションより下位の部分が各社ほぼ同一の分類方法になっていることが分かる。

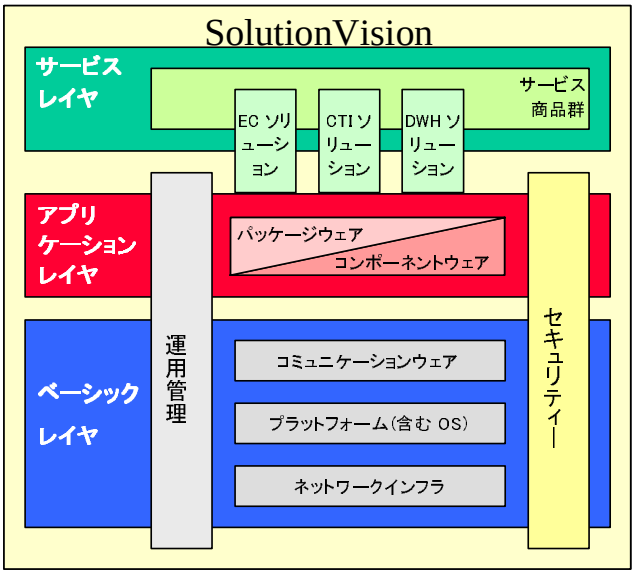


図3-9　SolutionVision

1998年1月には、東芝が同様なフレームワークをCSolutionを発表し[81]、同年4月には、日立がNetwork Objectplazaを発表した[74]。

今後に期待される新しいJavaベースのフレームワークとして、1998年3月にサンマイクロシステムズ社がEJB(Enterprise Java beans)を発表した[82]。EJBは、多階層（3階

層）の分散オブジェクト・アーキテクチャに基づいた Java 用のサーバ・コンポーネント・モデルである。今後の分散アーキテクチャの一つとして注目されている。

1999 年に入ると、インターネット時代のビジネスの流れを形成するのは、IBM 社であるとばかり"e ビジネス(e-business)"と言う言葉を大々的に宣伝し始めた[18]。一説に寄れば同社の年間 1,000 億円の広告宣伝費の内 250 億円も e ビジネスのイメージ戦略に使用したと言われている。インターネットの爆発的な普及に相前後して、各社はインターネット時代のビジネスに対応するためのソリューション・フレームワーク（体系）を続々と発表している。

1998 年 8 月に日立が Cyberfront を発表し[84]、1999 年 4 月に日本ユニシスが OnNet Solution を発表し[19,36]、同年 5 月には HP 社が E-service を発表し[13]、同年 7 月に富士通が Internet ビジネス・ソリューションを発表し[11]、同年 11 月には日本電気が iBestSolutions を発表している[98]。

図 3-10 に筆者らが開発した OnNet Solution の全体像を示す。情報基盤のレイヤー、アプリケーションのレイヤー、それらを統合してのサービスのレイヤーを定義している。

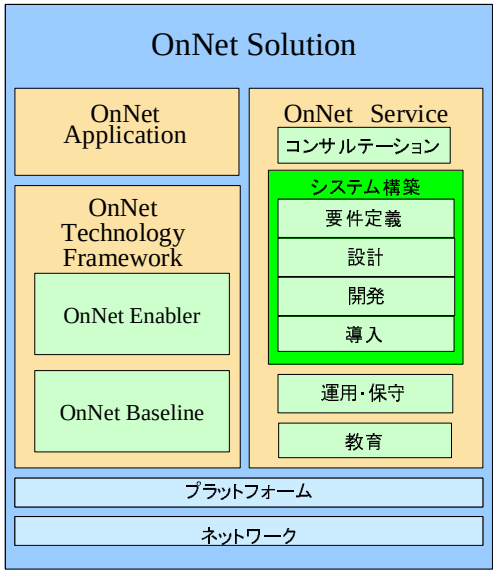


図 3-10　OnNet Solution

3.5 21 世紀、よりビジネス指向へ

各社より一通りソリューション・フレームワークが発表され、フレームワークに従ったソリューションが準備されようとしている。しかし、コンピュータがビジネスの道具であるという事実と、インターネットに代表される新しい通信インフラの整備により、このインターネットを有効に活用して、如何にビジネス活動を進めていくかに注力が注がれ始めた。各企業とも新しい時代のビジネス・モデルに注力を注いでおり、各社のソリューション・フレームワークの中にインターネットをベースにした新しいビジネス・モデルを投入し始めた。やがて、e ビジネスが常識になり、企業活動を表す「ビジネス」が e ビジネスと同義語になると予想している。

ユビキュタス・コンピューティングとか IBM 社のパーベイシブ・コンピューティング(Pervasive Computing)[38]と言われるように、ビジネス活動が「何時でも何処でも何処からでも(any time, any place, any where)」になるのは、21 世紀の早い時点と予想している。これからの新しい e ビジネスと既存の資産を統合する EAI(Enterprise Application Integration)が 2000 年の流行になった。

2000 年に入るとコンピュータの Y2K 問題が終了し、企業のコンピュータを取り巻く環境は大きく e ビジネスに向けて走り始めた。その時の、新しいビジネス・フレームワークのキーワードは、顧客中心、透明性、スピードであると言われている。各社とも、21 世紀に入ると、新しいマーケティング戦略として、次世代のマーキテクチャであるビジネス・フレームワークを発表している。筆者らは、日本ユニシスをこの新しい波に乗せるために、e ビジネス時代のソリューション体系（Unisys e-@ction Solutions）を発表し、発展させている[37]。

3.6 まとめと今後の課題

1980 年代の IBM が主導するプロプラエタリなアーキテクチャの時代から、1990 年代始めの Unix 陣営、OMG、オープン・グループ（X/Open と OSF の合同）等のオープン・アーキテクチャの時代に移ったかに見えたが、1990 年代後半に入りマイクロソフト、IBM、オラクル等のプロプラエタリなアーキテクチャの時代へと逆戻りしているようにも見える。ユーザに見える環境は、嘗ての IBM のメインフレームに接続された端末から Windows9x/2000/NT に替わり、マイクロソフトがアーキテクチャの主導権を握り動かしている。国際標準に替わり世界標準と自らのプロプラエタリな仕様を位置付けている。これは 1980 年代の IBM の姿そのままである。主役は交代したが、歴史が繰り返えされている。

3. ソフトウェア・アーキテクチャの発展

1990 年代の中頃を考えてみると、インターネットがここまで普及すると予想できたであろうか。21 世紀が本格的に動き始めた現在、情報技術の発展のスピードはますます速くなり、全ての計算機が繋がることにより、新たな問題、例えばコンピュータ・ウイルスやサイバーテロ等、が発生している。何が起きるか予想することは困難である。このように激しく動く時代において、何を基準にすればよいのであろうか。

変わらないのは、企業にとってコンピュータやインターネットは、目的ではなく、あくまでも手段・道具であるという事実である。この道具を使用して新しいビジネスを始めるかも知れないし、商品自体がコンピュータであることもある。企業が必要とするのは、ビジネスを活性化する道具（ソリューション）や道具立て（ソリューション・イネーブリング）である。求められているのは、企業のビジネスを支援するサービス、その為の道具立てである。それはオープンな情報基盤の上に構築されたソリューションであったり、ソリューションを作成するツールであったり、プロプラエタリなアーキテクチャ上のソリューションであるかも知れない。それらの多くは、勝ち馬のプラットフォームや情報基盤上に作られたソリューションである。

そのようなソリューションを提供するためには、利用者の視点に立ったソリューションのモデルを導入し、そのためのソフトウェア・アーキテクチャや考え方（フレームワーク）を作成することが重要である。従来、コンピュータ業界におけるアーキテクチャやフレームワークは、コンピュータ・メーカのビジネス戦略の一貫としてのマーケティング戦略として作成され、メーカ側のビジネスを正当化するための道具として利用されてきた。それは、コンピュータ・ハードウェアやインフラストラクチャとなるソフトウェアを提供する側の理論であった。視点を変えて、利用者のソリューションの視点よりソフトウェア・アーキテクチャのあるべき姿を問い直す必要があると考えている。それが次世代ソフトウェア・アーキテクチャの設計思想の柱の一つとなるものである。

4. クライアント／サーバ・システムのフレームワーク

4.1 あらまし

1990 年代始めの米国経済の停滞期に始ったオープン化の本格的な波が数年遅れで日本でも始り、オープン化、クライアント／サーバ(C/S)システム、ダウンサイジング等のキーワードが書籍、新聞、雑誌を賑わした。また数多くのクライアント／サーバ用のソフトウェアが市場に氾濫した。1994 年時点での新聞見開きサイズの C/S 情報基盤の一覧表でさえ C/S システム関連のツールが 500 種類程列挙される[76]、一説では全世界で 7 千とも 1 万とも云われた。しかし、ユーザがこれらの多くのソフトウェアを活用してアプリケーションを構築しようとしても、使用するソフトウェア間での相互接続性や相互運用性、所謂ソフトウェア間の整合性検査に多くの時間と労力を費やすのがつねであり、日本ユニシスの C/S システムのシステム・インテグレーションの経験でも、整合性検査に多くの時間と労力を費やす事例が多かった。これは、当時の C/S システムの構築が発展途上の技術であり、標準に基づく安定したソフトウェア・ツールや確立した方法論がまだ存在していなかったためである。

C/S システムを構築するとき、そのシステムの企業情報システム全体における位置付けやその C/S システムの分散形態、そして採用するハードウェア・プラットフォームやミドルソフトウェアを含んだ情報基盤および開発・実行・運用のためのソフトウェアの選択が重要になる。

企業情報システムの分類として、従来、金融業界に於ける勘定系、情報系、OA 系などの分類が著名であるが、先進企業は新しい分類を試みている。取り扱う情報の性格によって「対顧客系」、「対オフィス系」そして「対マーケット系」に分類してシステムの見直しを考えたり[62]、「基幹系業務」、「情報利用系業務」そして「身の回り業務」に体系化し、システムの方針を明確にしようとしている企業もある。生の情報をリアルタイムでオンライン・トランザクション処理する基幹系情報システム、基幹系情報システムを補完する情報系システムと経営的判断をする管理系システムを含む支援系情報システムに大別することを提案している企業もある[24]。またビジネスの将来像として競争相手、顧客、関連企業、供給業者をも結んだ企業間コンピューティングの重要性が指摘されている[85]。

C/S システムの分散形態に関しては、有名な例として、ガートナ・グループの分散コンピューティング・モデルがある（第 5.6 節の図 5-22 参照）。このモデルでは、ネットワークの所在場所によって、分散表示、遠隔表示、分散機能、遠隔データアクセスと分散データ処理の五つに分類している。分散表示、遠隔表示、分散ビジネス・ロジック、遠隔デー

タ管理そして分散データ管理に分類した例や[6]、また代表的な顧客のアプリケーションをサーベイし、分散システムを、遠隔表示型、フロントエンディング型、分散ロジック型、データステージング型、資源集中型、プロセス駆動型と複数アプリケーション型の七つの型に分類した例もある[79]。

第３章で述べたように、情報基盤の情報技術のフレームワーク（体系）は、1987 年のIBM 社の SAA で始まったプロプラエタリなフレームワークで始まり、1990 年に入りユニシス社より発表されたUnisys Architecture[87,95]や1994 年に発表されたIBM 社のOpen Blueprint[15]のような国際標準や業界標準ベースのオープンなフレームワークへと発展している。Posix OSE[23]が 1996 年 12 月に国際標準になったが、技術体系として普及していないのが現状である。

本章では、筆者が 1996 年に開発した C/S システム構築のためのフレームワークを解説する。次章では、このフレームワークを発展整備させたソフトウェア・アーキテクチャを論じる。この二つの章はお互いに関連しており、本章はいわば 1990 年代前半のビジネス環境を基盤としているが、第５章ではさらに新しい環境を対象としている。まず、フレームワーク（「オープン・ソリューション・フレームワーク(OSFW)」）の全体像を説明する。次に、フレームワークの構成要素である、ユーザの企業情報システムの一般的なモデル(「企業情報システム・モデル」）と C/S システムの情報基盤のための情報技術体系（「クライアント／サーバ・ソリューション（C/SS）フレームワーク」）を説明する。C/S システムのクライアント／サーバ・ソリューション（C/SS）モデルを説明し、これら C/SS モデルを構築・実行・運用そして他システムとの接続を支援する時に使用するプロダクトの組み合わせパターン（「オープンソフトウェア・プロダクトセット」）を示し、推奨するプロダクトの組み合わせを例示する。最後に、アプリケーション・システムのインフラストラクチャの選択への適用事例を示す。

4.2 オープン・ソリューション・フレームワークの概要

C/S システムの構築には、図 4-1 に示すように大きく三つの工程がある。

１）ビジネス戦略計画：情報戦略立案、解決策定義、BPR、プロジェクト定義
２）情報ソリューション計画：要求定義、アーキテクチャ定義、システム基本設計
３）ソリューション構築：詳細設計、開発、プラットフォーム／ソフト導入、テスト

OSFW は、上記２）３）の設計・構築段階のガイドとなるものである。これを参照することにより、従来多くの時間と労力を費やし、かつ品質面でも問題の多かった C/S システムの構築をより迅速にかつ低コストで、高品質に実現できる。

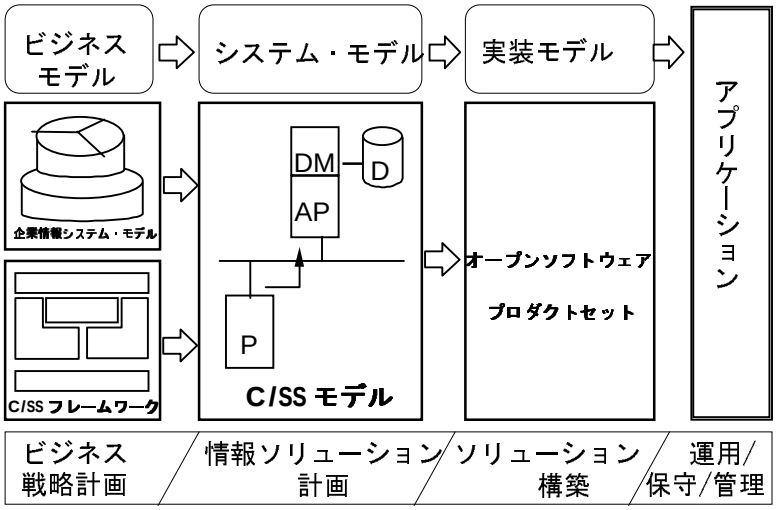


図 4-1　C/S システム構築の流れ

OSFW の設計に際して、日本市場への UA での経験を生かして、次の設計指針を設定した。我々は、これを ABCDEF コンセプトと呼んでいる。

- Available Information Technology（利用可能な情報技術を利用する）
- Business Application（フレームワークの対象は、ビジネス・アプリケーション）
- Client/Server Model（クライアント・サーバ処理形態を対象にする）
- Distributed Computing Model（分散処理形態も対象に含める）
- Easy to understand（まず、理解しやすいフレームワークであること）
- For our customers（フレームワークが我々のお客様に役立つこと）

端的に述べると、現在利用可能な情報技術を利用して、利用者が理解し易いフレームワークを提供することを目標にした。これは、数年先に利用可能になる情報技術を前提にした従来のソフトウェア・アーキテクチャを導入したことでの反省をベースにしている。早い技術進歩やビジネス展開に追随するための方策の一つでもあった。

OSFW は、Unix や PC のみならず、エンタープライズ・サーバを含むプラットフォームを使用してクライアント／サーバ・ソリューションを実現するためのガイドライン（構築体系）であり、(1)企業情報システム・モデル、(2)C/SS フレームワーク、(3)C/SS モデルと、それらに基づくプロダクトやサービスなどから構成される。オープン・プロダクトとしては、(4)オープンソフトウェア・プロダクトセットを定義している。これら四つのガイドラインは密接に関連している。

4.3 企業情報システム・モデル

オープン･ソリューション・フレームワークでは、企業の情報システムを、目的、役割、および取り扱う情報の視点から**業務系**、**情報系**、**オフィス支援系**に大きく分類する[50]。そしてこれらの系内、系間および外部間で情報共有と連携を支援するための**連携基盤**を明確化した。企業の情報システムが連携する外部としては、関連企業、供給業者、調査機関、金融機関、顧客、競争相手が存在している。そして、外部との連携は増大し、ますます重要なものになると考えている。関連を図 4-2 に示す。

尚、本章で提示しているモデルは、「誰のための情報システムか」という観点より見ると理解しやすい。旧来の呼び方および情報と本章におけるモデルとの比較を表 4-1 に示す。

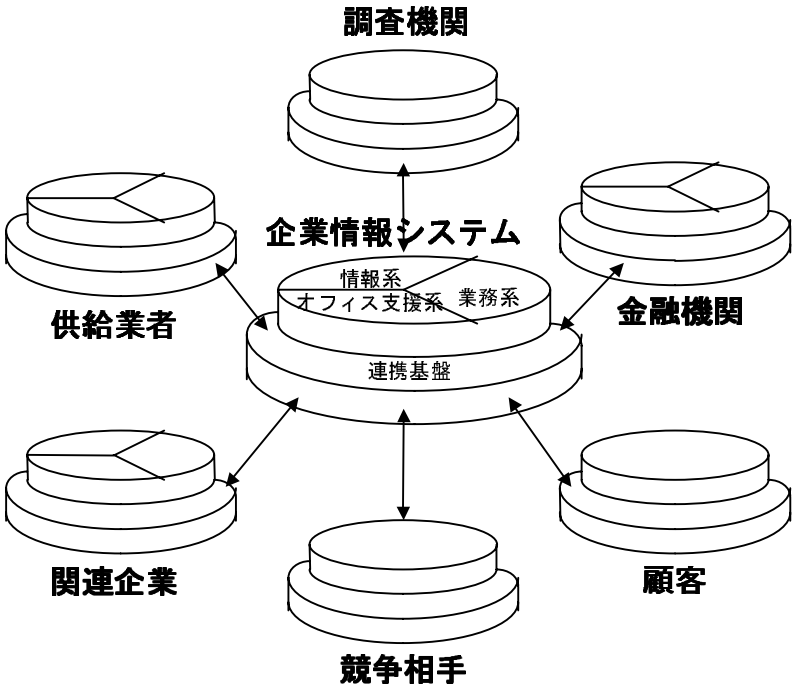


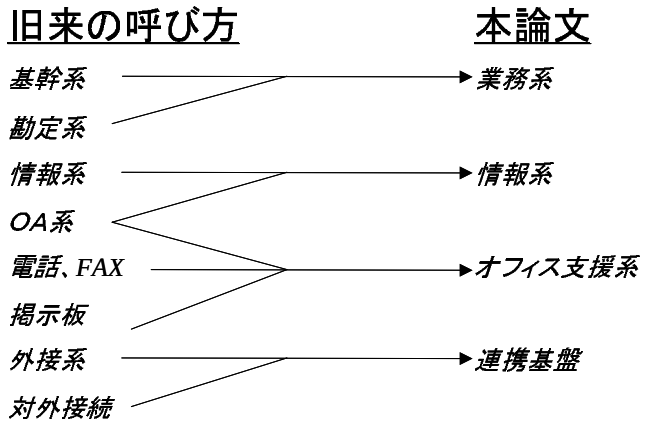
図 4-2　企業情報システム関連図

企業情報システムで使用される情報を次に示す。これら情報と各系間の関連を図 4-3 で示す。

- ■ 商品情報：　注文、約定、決済、開発状況、物流状況
- ■ 顧客情報：　販売記録、クレーム記録、一般会社情報（資本金、業績、取引先など）
- ■ 経営情報：　人事情報、財務情報、経理情報、統計情報
- ■ 市場情報：　競合他社情報、公共情報、社会情報

■ オフィス情報：文章情報（報告書等)、文書管理情報、伝言メッセージ

表 4-1　モデル名の対応



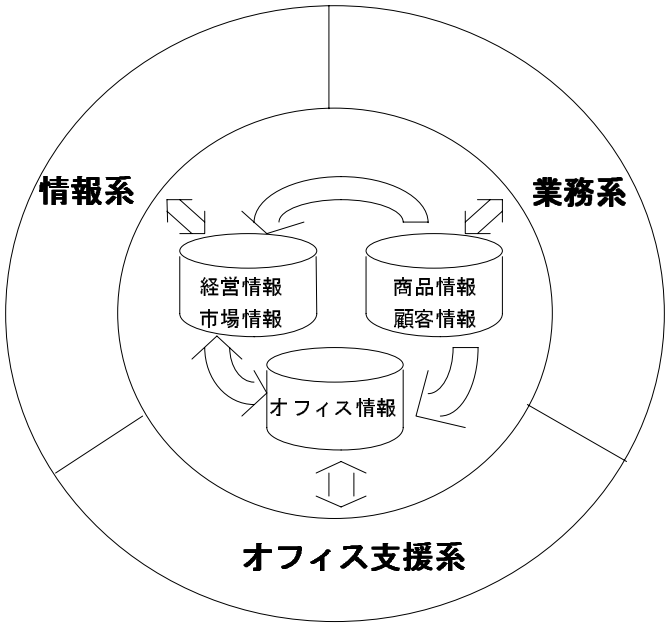


図 4-3 企業情報システム

企業情報システム・モデルの三つの系および連携基盤の目的、対象業務、主要機能、および主要アプリケーションを表 4-2 に示す。

表 4-2　三つの系と一つの基盤

	業務系	情報系	オフィス支援系	連携基盤
目的	日常業務の基本プロセスの自動化	情報の活用による企業活動の改善	オフィスワークの効率改善	企業内および他企業との情報の共有と連携の迅速化
対象業務	企業活動の基盤となる業務でその業務が停止すると企業活動そのものが停止するおそれのある業務	検索／加工／分析／予測／統計解析／シミュレーションなどの計画や意志決定を行うような業務	保管／共用／再利用／配布／伝達などの業務を補完・促進するような処理。従来、ワープロ文書、E-Mail、掲示板、FAX、Tel 等で流される情報の共有が含まれる。	日常業務の基本プロセスの自動化
主要機能	・定型的トランザクション処理と決まった形式の帳票などの定期的発行 ・商品情報・顧客情報の生成・蓄積・更新	・エンドユーザに検索と分析の機能を提供 ・業務系の生み出した情報および経営情報と市場情報を加工・利用	・個人の情報処理能力の拡大、およびオフィス業務の生産性向上 ・オフィス情報の生成・加工・共有・流通	商品情報、市場情報、オフィス情報、公共情報等を流通させるサービスを提供
主要なアプリケーション	・基幹アプリケーション －座席予約システム －営業情報システム －生産管理システム －物流システム －勘定系システム ・業務支援アプリケーション －人事システム －経理システム	・経営者支援システム ・営業支援システム ・管理者支援システム ・意志決定支援システム ・戦略情報システム ・エンドユーザ・コンピューティング	・グループウェア ・ワークフロー ・オフィス情報システム	・ネットワーク・セキュリティ（Firewall 等） ・部門間／企業間ワークフロー ・業務アプリケーション連携

企業情報システム・モデルの各系を実現するためには、システムの目的、業種、規模等によって各種の C/SS モデルが組み合わされて使用される。各系で典型的に使用されるモデルを表 4-3 に示す。

表 4-3　企業情報システム・モデルと C/SS モデル

企業情報システム・モデル	C/SS モデル
業務系	集中トランザクション処理モデル 分散トランザクション処理モデル
情報系	リモート・データ・アクセス処理モデル 分散データ・アクセス処理モデル データ・ステージング処理モデル
オフィス支援系	集中メッセージ処理モデル 連鎖型メッセージ処理モデル
連携基盤	データ・ステージング処理モデル 連鎖型メッセージ処理モデル

4.4 クライアント／サーバ・ソリューション・フレームワーク

C/SS フレームワークの目的は、C/S システムの実行時の情報基盤に技術体系を与えることである。C/SS フレームワークでは、ソフトウェアを五つのサービス・セット、**開発サービス**、**クライアント実行サービス**、**サーバ共用サービス**、**分散基盤サービス**と**管理サービス**、に分類している。サービスとは、ソフトウェア中に実現される代表的な技術のビルディング・ブロックである。

図 4-4 で C/SS フレームワークの全体構造を示す。

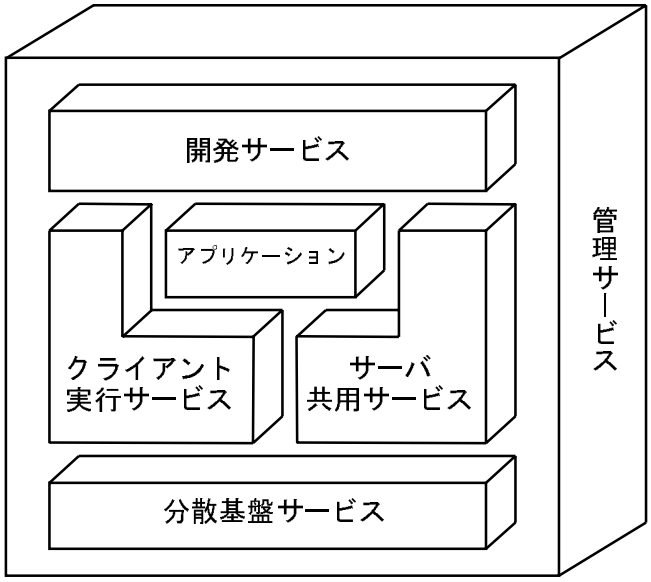


図 4-4　C/SS フレームワーク

4.4.1 開発サービス

開発サービスは、五つのサブサービスにさらに分類する。それらは、ユーザ・インタフェースの構築を支援する**ユーザ・インタフェース開発サービス**、3GL（C、COBOL、等）による **3GL 開発サービス**、**4GL 開発サービス**、オブジェクト指向言語による **OOPL 開発サービス**、開発工程全般を支援する**統合開発サービス**、そして既存システムを新しい形式のシステムへと再構築する**リエンジニアリング・サービス**である。図 4-5 にその構造モデルを示す。

4.4.2 アプリケーション

アプリケーションは、二つのサブセットにさらに分類する。それらは、開発サービスによって作成された**ユーザ作成**のアプリケーションと市販の**既製**のアプリケーション・パッケージである。図 4-6 にその構造モデルを示す。

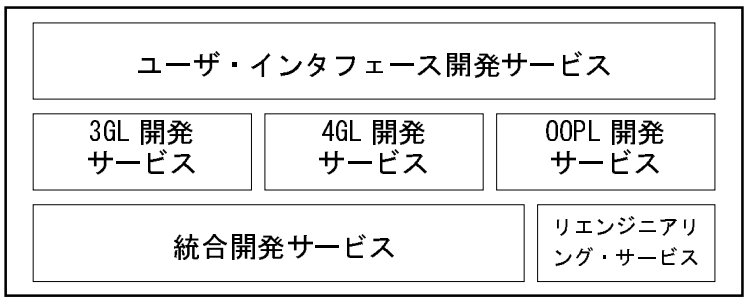


図 4-5　開発サービス

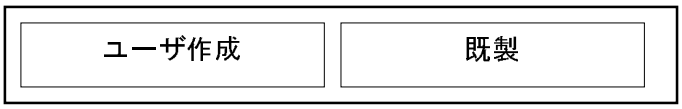


図 4-6　アプリケーション

4.4.3 クライアント実行サービス

クライアント実行サービスは、五つのサブセットにさらに分類する。それらは、ユーザとアプリケーション間の対話を実現する**プレゼンテーション・サービス**、ユーザにパーソナル・オフィス処理機能を提供する**デスクトップ・サービス**、エンドユーザ主導型で、対話的な情報活用を支援する **EUC サービス**、ワークフローに関与する個人に作業プロセスの自動化機能を提供する**ワークフロー・サービス**、そして遠隔データをアクセスする API を提供する**情報アクセス・サービス**である。図 4-7 でその構造モデルを示す。

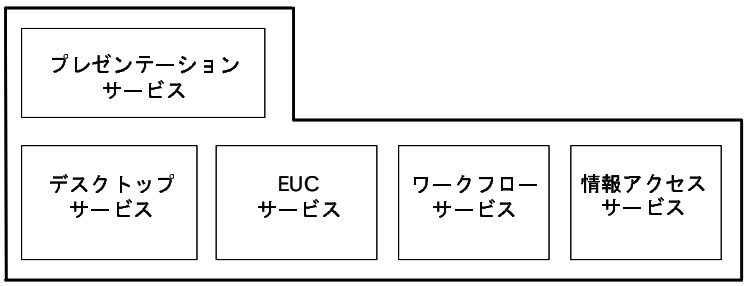


図 4-7　クライアント実行サービス

4.4.4 サーバ共用サービス

サーバ共用サービスは、五つのサブセットにさらに分類する。それらは、ネットワーク全体にわたる情報の確実な格納・共有・検索・操作を提供する**情報管理サービス**、デスク

トップ･サービスと協調して複数のオフィス･ユーザを支援する統合環境を提供する**オフィス共用サービス**、ＥＵＣサービスと協調して複数のエンドユーザに情報活用のための統合環境を提供する**EUC 共用サービス**、ワークフロー・サービスと協調して組織またはグループに作業プロセスの自動化機能を提供する**ワークフロー管理サービス**、そしてトランザクションの確実な実行と制御を提供する**トランザクション処理サービス**である。図 4-8 にその構造モデルを示す。

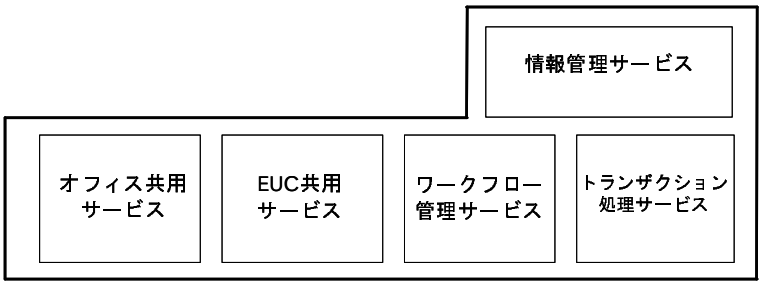


図 4-8 サーバ共用サービス

4.4.5 分散基盤サービス

分散基盤サービスは、二つのサブセットにさらに分類する。それらは、クライアントおよびサーバ間の相互運用性（インタオペラビリティ）を実現するための機能を定義する**分散協調サービス**とプラットフォーム間を各種の通信プロトコルで接続し、データ、テキスト、グラフィックス、音声などの情報の転送を行う**相互接続サービス**である。図 4-9 にその構造モデルを示す。

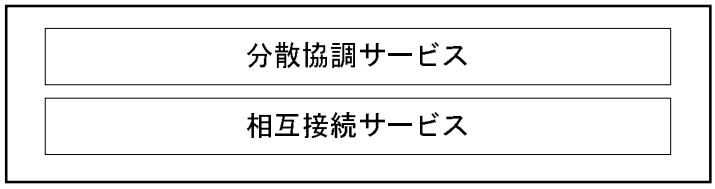


図 4-9 分散基盤サービス

さらに、分散協調サービスは、10 個のサブセットに分類している。それらは、分散システム環境で独立したプロセス間の通信と同期調整を行う**プロセス間通信**、協定世界時を提供する**タイム・サービス**、オブジェクト情報の管理のための**ディレクトリ／ネーミング・サービス**、ユーザ認証やメッセージ暗号化および暗号キー管理する**セキュリティ・サービス**、情報ネットワーク内に所在するオブジェクトに関してオブジェクト・リンキングとオブジェクト・リクエスト・ブローカ（ＯＲＢ）機能といったサービスを提供する**オブジェクト・サービス**、グローバル・トランザクション処理やトランザクション転送を行う**分散**

トランザクション処理、ネットワーク経由のデータ、イメージ、音声などの情報交換のための**メッセージ通信処理**、ネットワーク経由のアプリケーション間のデータ、テキスト、グラフィックス転送のための**ドキュメント交換**、ネットワーク経由のファイル転送、分散ファイル・アクセスと分散プリントを行う**ファイル／プリント・サービス**、そして他のデータベース管理システムとの協調処理のための**協調データベース・サービス**である。図 4-10 でその構造モデルを示す。

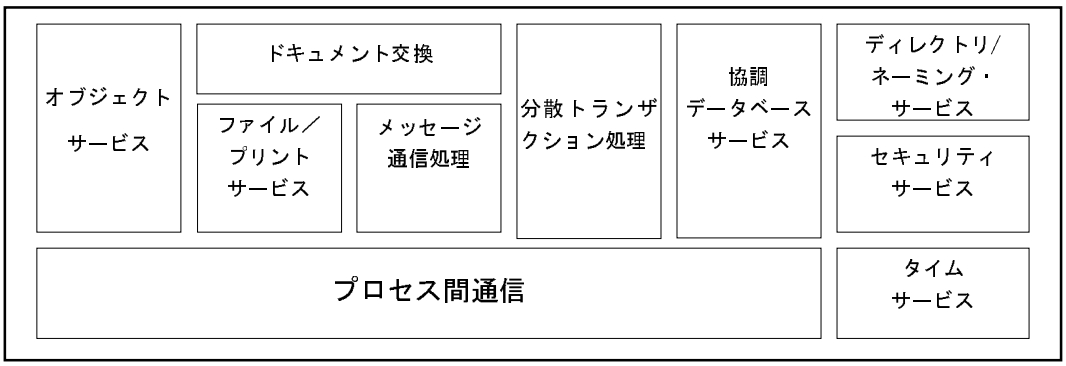


図 4-10　分散協調サービス

相互接続サービスは、OSI 参照モデルの第 3 層と第 4 層に対応する**データ転送サービス**と OSI 参照モデルの第 1 層と第 2 層に対応する**サブネットワーク・サービス**に分類する。

データ転送サービスには、OSI データ転送、TCP/IP データ転送、DCA データ転送、BNA データ転送、SNA データ転送、Novell NetWare データ転送、そして LAN Manager データ転送が含まれる。サブネットワーク・サービスには、LAN サブネットワーク・サービスと WAN サブネットワーク・サービスが含まれる。図 4-11 でその構造モデルを示す。

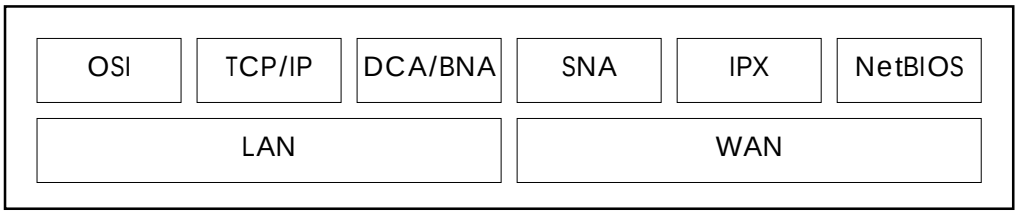


図 4-11 相互接続サービス

4.4.6 管理サービス

管理サービスは、六つの管理機能を中心とし、三つのインタフェースとそれらを支える管理基盤から構成される。

六つの管理機能は、ハードウェア／ソフトウェアの構成、情報管理、配布、適用を支援

する**構成管理機能**、障害発生の監視、報告支援と予防保守を支援する**障害管理機能**、管理対象の定義、監視、分析／評価および報告支援とキャパシティ・プランニングを支援する**性能管理機能**、使用実績収集、計算、報告（請求）と制御を支援する**課金管理機能**、認証／アクセスに関する登録、監視と制御を支援する**セキュリティ機能**、そしてコンピュータ・システムの運用と操作を支援する**運用・操作支援機能**である。

三つのインタフェースは、管理情報の表示と管理用コマンドの入力処理を支援する**管理者インタフェース**、開発環境との連携を図るためのインタフェースとユーザ固有管理アプリケーションの構築支援ツール群を支援する**開発環境インタフェース**、そして実行環境との連携を図るための**実行環境インタフェース**である。

管理基盤は、オブジェクト指向管理基盤によって実現された管理情報ベースと管理情報の伝達／保持の仕組みである。

図 4-12 にその構造モデルを示す。

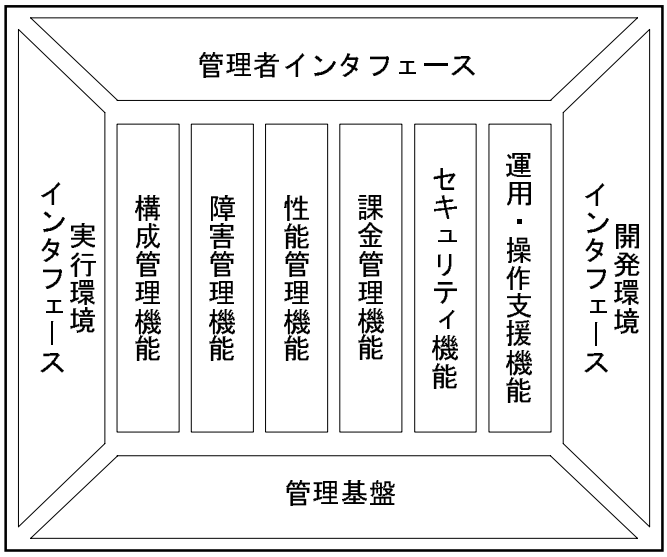


図 4-12 管理サービス

4.5 クライアント／サーバ・ソリューション・モデル

C/SS モデルは、データの保管場所の視点およびクライアントとサーバ間の処理形態の視点より表 4-4 のような七つの「基本モデル」からなる。この分類は、通常のクライアント／サーバ・モデルに対して時間の概念および垂直・水平分散の概念を導入したことになっている。現実の C/S システムは、これらの基本モデルの一つ以上の組み合わせで構成される。[40]

表 4-4　C/SS モデル

データ形態 / 処理形態	集中	分散	
		垂直	水平
トランザクション処理	集中トランザクション処理モデル	分散トランザクション処理モデル	
依頼応答処理	リモート・データ処理モデル	分散データ処理モデル	
遅延処理	集中メッセージ処理モデル	データ・ステージング処理モデル	連鎖型メッセージ処理モデル

このモデルは、次章で九つのアーキテクチャ・スタイルとして発展させている。従って、本論文では、各モデルの詳細は省略する。

4.6 オープンソフトウェア・プロダクトセット

C/S システムの実行形態をモデル化した C/SS モデルに従って、C/S システムの開発・実行・管理そして他システムとの接続を支援するためのプロダクトセットを記述する。

プロダクトセットは、七つの C/SS モデルに対応した C/S システムを開発・実行するための 7 つの基本パターン（図 4-13）と運用管理のための三つの基本パターン（図 4-14）および接続のために三つの基本パターン（図 4-15）からなる。現実の C/S システムは、これら基本パターンの一つ以上の組み合わせで開発・実行・管理されそして他システムと接続されている。

それぞれに推奨できる市販のソフトウェア・プロダクトの組み合わせがある。第 6 版（1997 年 11 月版）では、55 種類のセットがあった。ここではその全てを紹介できないが、簡単なセットだけを紹介する。これらのプロダクトセットは、一定の選定基準を設け、それに従い社内に評価チームを結成して十分な評価を実施してから選考したものである。

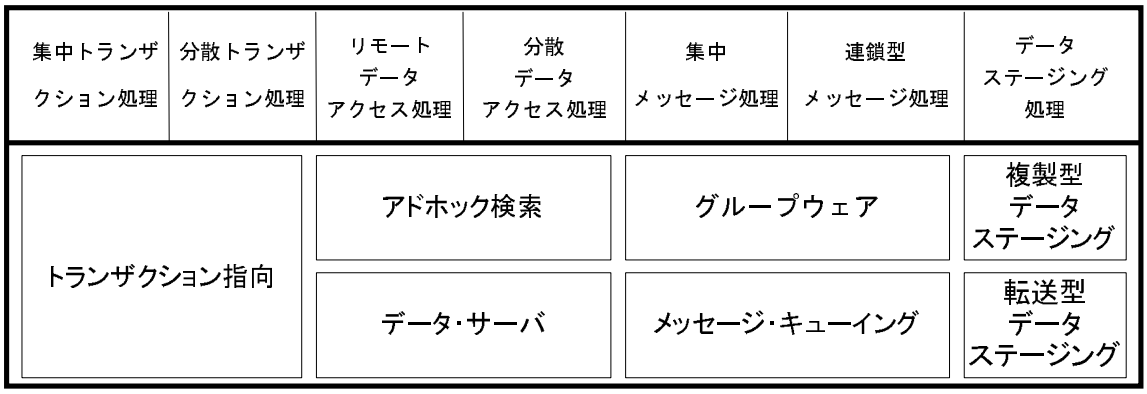


図 4-13 構築の基本パターン

図 4-14 管理の基本パターン

図 4-15　接続の基本パターン

4.6.1 トランザクション指向タイプ

C/SS モデルの「集中トランザクション処理モデル」および「分散トランザクション処理モデル」の構築であるトランザクション指向パターンは、アプリケーションの実行がサーバ中心に行われるものとクライアントとサーバ全体として処理を行っていくものとに分類する。

１）サーバ中心処理
トランザクション件数が比較的少なくサーバ処理が重いタイプ（例：Unix LINCⅡ ＋ Oracle）

２）会話中心処理
トランザクション件数が多く、短い応答が要求されるタイプ（例：Visual Basic + BEA Tuxedo + Oracle）

4.6.2 アドホック検索タイプ

必要な検索条件を指定して、サーバのデータベースからデータを表計算のシートに取り込み、データ分析、レポート作成等を行う。アプリケーションの開発を伴わないアドホックな検索タイプである。一般に非定型処理である。
（例：EXCEL + Excellent + Oracle）

4.6.3 データ・サーバ・タイプ

データベースをアクセスする業務アプリケーションを開発するためのツールとミドルソフトウェアの組み合わせ。以下の三つのタイプに分けてソフトウェアを選択する。一般に定型処理である。

１）規模：小、機能：検索中心（例：ACCESS + SQL*NET + Oracle）
２）規模：中、機能：検索・簡易更新（例：Visual BASIC + SQL*NET + Oracle）
３）規模：大、機能：検索・更新（例：PowerBuilder + SQL*NET + Oracle）

4.6.4 グループウェア・タイプ

グループウェア・タイプは、アプリケーションの開発が伴わない電子メール型とアプリケーションの開発が伴う情報共有型とワークフロー型に分類する。

１）電子メール型（例：MS-Exchange）
２）情報共有型（例：Lotus Notes）
３）ワークフロー型（例：Staffware）
４）ドキュメント管理型（例：Interleaf）

4.6.5 メッセージ・キューイング・タイプ

分散システムにおいて非同期に処理を連携するときに使用する。（例：MQseries）

4.6.6 複製型データ・ステージング・タイプ

複製（レプリケーション）機能を利用してデータ・ステージングを行う。

4.6.7 転送型データ・ステージング・タイプ

ファイル転送機能を利用してデータ・ステージングを行う。

4.6.8 管理の基本タイプ

運用管理では、サーバ（S）数とクライアント（C）数を目安にシステム規模を大きく三つに分類してプロダクトセットを設定している。

1) 小規模管理型：S＝1～2、C＝30 以下、クライアント管理、印書管理を行う。
2) 中規模管理型：S＝3～10、C＝31～300、集中管理を行う。
3) 大規模管理型：S＝11 以上、C＝301 以上の拠点での管理とそれを統合管理する階層型管理を行う。

4.6.9 接続の基本タイプ

接続用のプロダクトセットは、接続の形態により次の三つに分類する。

1) インターネット接続型
   インターネットを利用して情報を発信、収集、交換する。便利なフリーソフトがあるのでこれを利用する。
2) オープン接続型
   マルチベンダのホスト・コンピュータを接続する TCP/IP 接続やファイル転送と端末エミュレーション機能が一般的である。
3) メインフレーム接続型
   メインフレームとの接続を高いレベルのプロトコルまでサポートする。

4.7 アプリケーション・システムのインフラストラクチャ選択への適用

C/SS モデルは、顧客のアプリケーション・システムのインフラストラクチャ（情報基盤）を選択する手法の一部としても使用する。顧客の新しいシステム開発のコンサルテーション時に、ITIP(Information Technology Infrastructure Planning)方法論を適用したので[43]、その適用概要を論じる。

4.7.1 S 建設会社のシステム要件

S 建設会社は、国内は北海道から沖縄まで、さらに海外においても建設事業を展開する総合建設事業者であり、今回の新財務システムの展開範囲は、国内建設事務所（約 2,000 箇所）を想定している。

建設事業の特性として展開対象の建設事務所は全て、工事発注に伴い設営され、工事完了に伴い取り壊される仮設事務所であり、建築主体の S 建築会社は年間約半数の仮設事務

所が、設営、取り壊しを繰り返す。従って、システム展開によって設営されるパソコンも同様に年間約 1,000 箇所での移設が繰り返されることになる。

建設事務所を束ねる役割で、一県１～３カ所程度の常設営業所が存在しており、更にそれらを束ねる支店（会計単位）が、全国 14 カ所に存在し、本社を加えて会計業務機能を遂行している。

建設業は、基本的に現地搬入、現地精算を基本とする為、鉄骨、生コンを始めとした資材や、鳶職、大工などの労務費から、作業所運営の為の経費まで全て作業所で査定し、支払申込手続きが行われる。同様に、その作業所で発生する契約、受注報告、各種振替等等総ての会計手続きも現地作業所で入力する。

(1) 作業所の環境

- 現在は総ての会計手続きを伝票起票で行っている。
- 新財務システム展開後の入力予定者は、基本的に女性事務担当者であり、会計業務、機器操作共に、プロでない人間であり、新財務システム展開により初めて機器操作を行うものも多い。
- 機器設置に関しても、同様に作業所独自では実施出来る能力はなく、さらに仮設事務所である事、事務所内の配置変えが頻繁に起こる事からも、ネットワークの安定性は低い。
- 特にダム工事の作業所等、遠隔地の場合は故障時の対応等に問題がある。

(2) 営業所の環境

- 常設の営業所であり、管轄工事事務所から送られてくるデータを承認、査定し、支店にデータを送る。
- 基本的に県庁所在地等の都市部に存在しているが、数人単位が多く、機器運用、保守に関しては作業所と同じレベルである。

(3) 支店の環境

- 常設の事務所である。大規模、小規模の違いがあるが、管轄の作業所総てからデータが送信されてくる。
- 支店内に、コンピュータ保守要員として、情報システム部門からの出向者が１名在籍している。

4.7.2 C/SS モデルの活用

推奨する C/SS モデルを選択するために情報処理基盤構築のためのコンサルテーション手法である ITIP 方法論が採用している手順は、次の通りである。

1）データとその管理組織を明確にする。
2）必要なビジネス・プロセスを列挙する。
3）各データを分散配置した場合、集中配置した場合に想定される物理的配置を記述する。両方の記述が必要なのは、コンピュータ・システムに馴染みのない利用者部門の人達に提案するソリューションを理解して貰うためである。
4-1) データを集中配置したときの各ビジネス・プロセスの C/SS モデルを明確にする。
4-2) データを分散配置したときの各ビジネス・プロセスの C/SS モデルを明確にする。
4）運用管理、セキュリティ、ビジネス要件の観点からデータの配置を集中か分散かを決める。
5）将来性と安定性を考慮して、よりシンプルなモデルを選択する。

4.7.3 適用事例：新財務システム

この事例は、S 建設会社の新財務システムのために C/SS モデルをどのように導き出したかを論じる。導出の中心的な箇所のみを説明する。

1）財務システムの業務運用を組織と主要データの観点から整理する。データには、エントリィ・データ、全社共通データ等が含まれる。組織には、作業所、営業所、支店、そして本社がある。データと組織間の関連も整理する。
2）次の業務処理が選択される。
- データ・エントリィ処理
- 支店内処理（作業所経理処理、内勤経理処理、月次決算処理）
- 支店間処理（振替処理）
- 全社処理（決算処理）
- 周辺処理（マスターメンテナンス他）

3）データを、分散配置した場合、集中配置した場合を明確にする。まず管理主体からみたデータの関係を図 4-16 に示す。使用されるデータは、次の通りである。
- 全社データと全社共通データ（各種マスター）は集中
- 各支店データは水平分散
- 全社データは、全社共通データと支店データは垂直分散
- 転送前エントリィ・データは水平分散
- 支店データと転送前エントリィ・データは垂直分散

図 4-16 における各データを、分散配置した場合、集中配置した場合に想定される物理的配置を表 4-5 に示す。

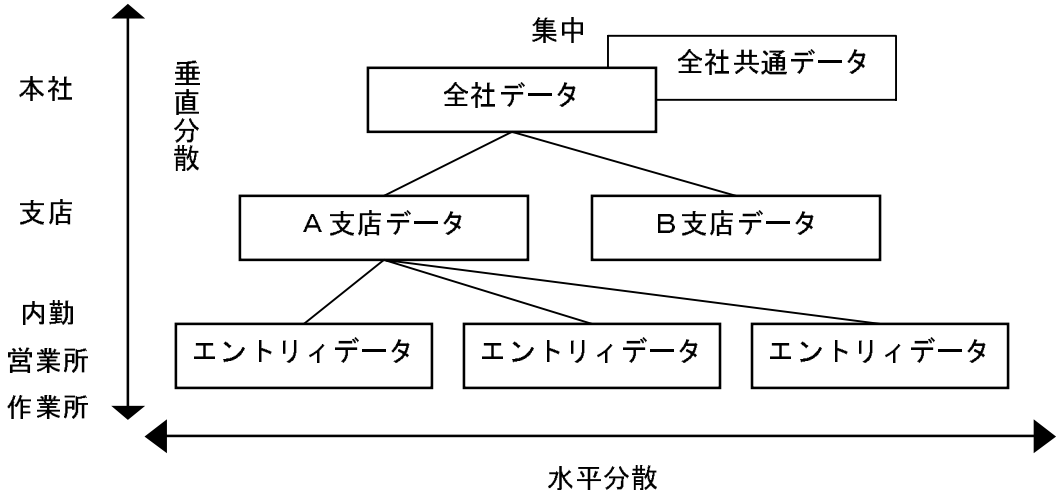


図 4-16　新財務システム・データ管理関係図

表 4-5　新財務システム・データ物理配置

	分　散　配　置	集　中　配　置
本社	全社データ 全社共通データ	全社データ 全社共通データ 転送後エントリィデータ
支店	支店データ 全社共通データ（コピー） 転送後エントリィデータ	
内勤 営業所 作業所	転送前エントリィデータ 全社共通データ（コピー）	転送前エントリィデータ

次に C/SS モデルに基づき、データを分散散配置した場合、集中配置した場合に分け、以下の前提の下に各処理がどの処理形態に該当するかを示す。

【前提１】 作業所に於けるクライアント環境は、常設性が低いという特殊性があり、常設ネットワークに接続された状態とするには困難がある。一方作業所に於けるデータ入力業務は、日々行われるものではないが、必要な時には使用可能であることが求められる。C/SS モデルの考え方では、このようなクライアント利用形態は、データ・ステージング処理モデルに含まれるモバイル・コンピューティングに最も近い。

【前提 2】 以下の処理形態の当てはめでは、トランザクション処理と依頼応答処理を以下の基準により適用している。

- クライアントとサーバ間にWAN（＝回線）がない場合－依頼応答処理
- クライアントとサーバ間にWANがある場合－トランザクション処理

【前提 3】 想定されるデータ量／トランザクション量からみて、全データを単一データベースとして保有することは可能と判断される。従って、本社にデータを集中配置する場合、単一データベースを前提とする。

（1）データを集中配置した場合の各処理のC/SSモデルを特定する。

- データ・エントリィ処理は、作業所／営業所／内勤でデータ入力が行われ、本社に転送され支店内処理に引き継がれる。データ配置が作業所／営業所／内勤と本社間で垂直分散であり、本社へのデータのアップロードが遅延処理である。従って、データ・エントリィ処理の該当するモデルは、データ・ステージング処理モデルである。
- 支店内処理は、本社に配置されたデータを対象とした、支店内端末からの問い合わせ応答処理である。処理形態は、支店と本社間にWANが存在するため、トランザクション処理である。従って、支店内処理の該当するモデルは、集中トランザクション処理モデルである。
- 支店間処理は、本社内に配置された二つの支店データを対象としたいずれかの支店端末からの問い合わせ応答処理である。データは、本社内に集中し、処理形態は、トランザクション処理である。支店間処理の該当するモデルは、集中トランザクション処理モデルである。
- 全社処理は、本社に配置されたデータを対象とした本社内端末からの問い合わせ応答処理である。処理形態は、WANが介在しないため依頼応答処理である。従って、全社処理の該当するモデルは、リモート・データ処理モデルである。
- マスターの登録／更新／削除処理等の周辺処理は、支店からの本社内に配置された全社共通データを対象とした即時処理である。データは集中で、処理形態はトランザクション処理である。従って、周辺処理の該当するモデルは、集中トランザクション処理モデルである。

（2）データを分散配置した場合の各処理のC/SSモデルも特定する。

- データ・エントリィ処理は、作業所／営業所／内勤でデータ入力が行われ、支店に転送される。データ配置は、作業所／営業所／内勤と支店間で垂直分散であり、支店へのデータのアップロードが遅延処理である。従って、データ・エントリィ処理の該当するモデルは、データ・ステージング処理モデルである。
- 支店内処理は、支店に配置されたデータを対象とした、支店内端末からの問い合わせ応答処理である。対象データは支店内に存在し、WANを介さないため、

依頼応答処理である。従って、支店内処理の該当するモデルは、リモート・データ処理モデルである。

- 支店間処理は、本社内に配置された二つの支店データを対象としたいずれかの支店端末からの問い合わせ応答処理である。対象データは、複数支店に水平分散して存在し、処理形態は、WAN を介するためトランザクション処理である。支店間処理の該当するモデルは、分散トランザクション処理モデルである。
- 全社処理は、本社に配置されたデータを対象とした本社内端末からの問い合わせ応答処理である。処理形態は、WAN が介在しないため依頼応答処理である。従って、全社処理の該当するモデルは、リモート・データ処理モデルである。
- 周辺処理は、以下の二つの処理に分けられる。

(1) 本社に配置された全社共通データの登録／更新／削除処理

マスターの登録／更新／削除処理は、支店からの本社内にある全社共通データを対象とした問い合わせ応答処理である。データは、本社内にあり、処理形態は、WAN を介するためトランザクション処理である。従って該当するモデルは、集中トランザクション処理モデルである。

(2) 本社から支店、営業所、作業所への全社共通データ配信処理

本社から支店、営業所、作業所への全社共通データの配信処理は、データが本社から支店、営業所、作業所と垂直分散し、処理形態は遅延処理である。従って該当するモデルは、データ・ステージング処理モデルである。

4）以下の要件により、新財務システムのデータの物理的配置は、集中型が最適と判断した。

- 現状以上の運用負荷を極力抑える。
- 作業所環境を考え、噴出・盗難など広い意味でのセキュリティを重視する。
- データは現場入力時にチェックを行う（クリーンデータ化）。

5）データを集中配置した場合の処理形態は、集中トランザクション処理モデル、リモート・データ処理モデルおよびデータ・ステージング処理モデルの組み合わせである。この内、リモート・データ処理モデルが適用されるのは、全社処理モデルの部分のみである。【前提 2】で記したようにリモート・データ処理の替わりにトランザクション処理を適用することは可能であり、業務要件からは、リモート・データ処理である必然性はない。

インフラから見た場合には、将来性・安定性を考えて、より単純なモデルほど望ましいことから、全社データ処理のリモート・データ処理モデルを集中トランザクション処理モデルに置き換える。

結論として、新財務システムのインフラ・モデルは、データ配置は本社集中、処理形態は以下の組み合わせとする。

- 常設性の低い全国 2,000 箇所の作業所での現場データ入力と本社への電子データ転送には、データ・ステージング処理モデルを適用する。
- 本社に集中されたデータを対象とした依頼応答処理には、集中トランザクション処理を適用する。

この分析をベースにして、各モデルに必要なソフトウェア・プロダクトの推奨する組み合わせと必要なハードウェア構成を提案した。運用管理用のプロダクトを除いて、各 C/SS モデル毎に提案したソフトウェア・プロダクトの組み合わせは、次の通りであった。

(1) 集中トランザクション処理モデル

クライアント・アプリケーション用に VisualBasic、サーバ上のトランザクション管理に BEA Tuxedo そしてデータベース・ソフトウェアとして Oracle を使用する。

(2) データ・ステージング処理モデル

Oracle モバイル・エージェント

次のハードウェア機器を選択した。

(1) サーバ　　Sun Ultra Enterprise 4000 と
　　COMPAC PC サーバ

(2) クライアント　　DOS/V PCs とモバイル機能

S 建設会社は、筆者らが提案した情報インフラを選択し、アプリケーション開発を実施した。この適用事例は、S 建設会社に対する有償のコンサルテーションであり、S 建設会社および S 建設会社が使用しているビジネス・コンサルタント会社からも高い評価を得た。

4.8 まとめと課題

本章で解説した OSFW は、明確なことであるが、全てのアプリケーション領域をカバーしていない。さらに、従来の情報システムで取り扱われ今後も重要であるバッチ処理や大量プリント処理は今回のフレームワークの中では明示的には扱われていない。また、発展途上のオブジェクト指向やエージェント指向、さらにモバイル・コンピューティングについても同様である。もっともクライアントのクライアントとも呼ばれているモバイル・コンピューティングや PDA は、クライアントの一部が移動体（子クライアント）として動き、適当なタイミングで親のクライアントに統合される。これは「データ・ステージング処理モデル」の一種と見なすことが出来るが、エージェント指向等と合せて考察する必要があるので今後の検討課題としていた。

これらの問題点を解決するため、C/SS モデルを見直し、次章で論じる九つのアーキテクチャ・スタイルを開発した。

5. 分散処理システムのアーキテクチャ・スタイル

5.1 あらまし

システム開発のアーキテクチャ設計段階におけるシステムのアーキテクチャの設計は、効率性、信頼性、保守性等の各種のソフトウェア特性に影響を与える重要な決定である。一般的にシステム・アーキテクトは、開発するシステムの利用者や開発者及び管理者等の関係者より様々な要件を集め、既存のアーキテクチャ・スタイルや過去の経験をベースにして、要件からアーキテクチャを設計している。開発者は、アーキテクトによって設計されたアーキテクチャを使ってシステムを具体化する。アーキテクチャは、技術面からもビジネス面からも、さらに社会面からも影響を受けているのである。そして、作成された新しいシステムは、更に将来のアーキテクチャに影響を与える技術環境、ビジネス環境そして社会環境に影響を与えている。Bass は、この環境からアーキテクチャとシステムへ、そして環境へと繰り返すサイクルをアーキテクチャ・ビジネス・サイクル(ABC)と呼んでいる[4]。

ABC を図 5-1 に示す。図中の吹き出し個所（特性軸、アーキテクチャ・スタイルおよび選択方式）は、筆者らの研究の位置付けを示すために追加した部分である。

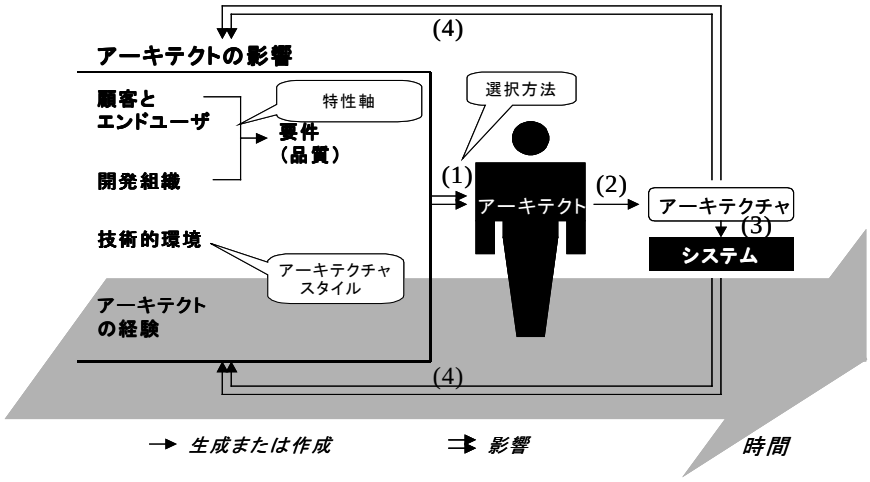


図 5-1　アーキテクチャ・ビジネス・サイクル

サイクルは次の通りである。

(1) アーキテクトは、要件や技術的環境及び経験をベースに、アプリケーション・システムのアーキテクチャを創出する。

(2) アプリケーション・システムのアーキテクチャを分析し生成する。

(3) アーキテクチャを使ってアプリケーション・システムを開発する。
(4) アーキテクチャ経験とアーキテクチャ・スタイルをフィードバックする。

筆者らは、クライアント／サーバ(C/S)処理を含んだ分散処理システムのソフトウェア・アーキテクチャ設計を支援するために、ソフトウェア・アーキテクチャの直感的な単純なモデルを作成した。このモデルを、1996 年にオープン・ソリューション・フレームワーク(OSFW)のクライアント／サーバ・ソリューション(C/SS)モデルとして発表した[40,41,43,45]。C/SS モデルにモバイルやエージェントのソフトウェア技術を取り込むために、モデルを再評価し、1999 年に分散処理システムのためのアーキテクチャ・スタイルを発表した[44]。このモデルでは、ビジネス・アプリケーションの分散処理システムを９つのアーキテクチャ・スタイルに分類した。

アーキテクチャ・スタイルそのものは、現場のシステム・エンジニアが直感的に理解でき、分散処理システムの参照アーキテクチャとして使用できるように設計している。しかし、システム・アーキテクトがアプリケーション・システムのアーキテクチャを決定しなければならない段階で、データの保管場所やクライアントとサーバ間やサーバ同士間の処理方式が決まっていないことは珍しいことではない。一般的にアーキテクトは、利用者や開発者や管理者の幅広い視点からアーキテクチャを決定している。従って、利用者や開発者の視点からでは適切でないが、管理者の視点から適切なアーキテクチャ・スタイルを選択することもある。我々は、経験の少ないアーキテクトのために単純で実践的なアーキテクチャ・スタイル選択手法の提供を要請されていた。

本章では、アーキテクチャ・スタイルを紹介し、アプリケーション・システム開発のためのアーキテクチャ・スタイル選択の単純で実践的な手法を提案する[46-48]。図 5-1 中の吹き出し個所は、ABC に於ける本章の研究の位置付けを示している。第 5.3 節で説明するアーキテクチャ・スタイルは、技術的環境に於ける参照モデルとして使用される。アーキテクチャ・スタイルの特性化は、要件を表現した特性軸である。そして、選択手法は、アプリケーション・システムの適切なアーキテクチャ・スタイルをどの様に選択するかのアーキテクトへの提案になっている。この選択手法は、第 5.4 節で説明する。この手法を、稼働中の実ビジネス・アプリケーションに適用した。その経験を第 5.5 節で説明する。最後に関連研究と結論を論じる。

5.2 クライアント／サーバ･モデルの拡張

本章で提案するアーキテクチャ･スタイルの説明に入る前に、スタイルの基本になった考え方を説明する。

本論文で対象とする情報処理システムは、主として企業におけるビジネス処理用のアプリケーションである。ビジネス処理を情報処理システムとして実現するために、ビジネス世界の商行為や商慣習をシステム化する。従って、「注文を出す」、「商品の在庫を調べる」、「商品を発送する」、「請求書を発行する」、「入金を確認する」等々のビジネスの手順やプロセスを抽象化することになる。

ビジネス行為をモデル化するために、まず、作業を依頼する「もの（物、者）」と作業を遂行する「もの（物、者）」、及びそれらの間の関連を抽象化する[88]。作業を依頼し、依頼に依って遂行し、結果を知らせるという一連の行為は、コンピュータ処理のモデルとしては、古典的な C/S モデルがよく適合している。しかし、現実のビジネスの進み方は、依頼しそれに同期して作業を遂行する作業形態だけでなく、依頼と依頼された作業の遂行が非同期に為されることが多々ある。従って、複数の作業が協調しあって作業を遂行する分散協調処理をも抽象化のモデルとして含む必要がある。

図 5-2 に依頼する「もの」と作業する「もの」の関連を示す。作業する「もの」は作業を遂行するために他に依頼することもある。

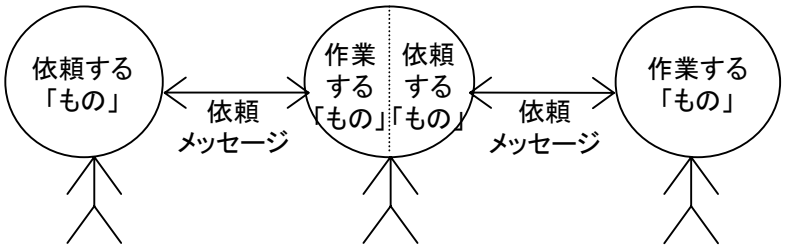


図 5-2　依頼する「もの」と作業する「もの」

一般に、作業の依頼メッセージには、結果を知る必要がある「作業依頼」や「問い合わせ」がある。これらは、依頼に対して同期的に作業を遂行し結果を報告することが要請されている。もう一つの依頼メッセージには、依頼した作業結果を知る必要がない、作業の「通達」がある。これらは、依頼に対して非同期的な作業実施が要請されている。従って、作業の依頼メッセージは、最小単位として作業依頼し結果を返すトランザクション型メッセージ、問い合わせに対して調査し応答する依頼応答型メッセージと依頼するだけの通達型メッセージに大別することが出来る。前者二つがクライアントとサーバが同期して作業する形態であり、後者が非同期で作業する形態である。

5.3 アーキテクチャ・スタイル

アーキテクチャ・スタイルの対象ドメインは、分散処理環境下のビジネス・アプリケーション・システムである。対象ドメインは、図 5-3 に示す通り情報ドメイン、ビジネス・ドメインとオフィス支援ドメインである[45]。情報ドメインは一般的にフロント・オフィスと呼ばれ、C/S 処理がその典型的なコンピューティング・パラダイムである。ビジネス・ドメインは一般的にバック・オフィスと呼ばれ、トランザクション処理がその典型的なコンピューティングパラダイムである。オフィス支援ドメインは、センター・オフィスとかミドルオフィスとか呼ばれ、グループウェア、ワークフロー、や電子メールのような協調処理がその典型的なコンピューティング・パラダイムである。

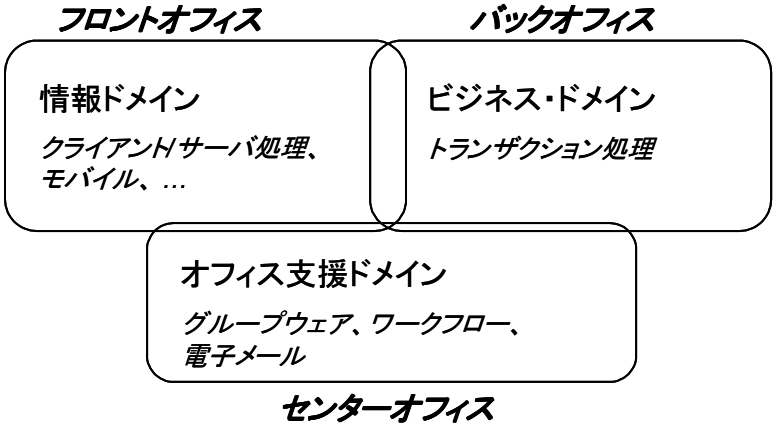


図 5-3 アーキテクチャ・スタイルの対象ドメイン

5.3.1 スタイルの分類

分散処理システムを次の視点で分類する。原則として、「プレゼンテーション(P)」は、クライアント側にあり、「データ(D)」は、サーバ側にあることを仮定している。

1) クライアントとサーバ間の処理方式が、クライアントの依頼に対してサーバが同期して処理する同期処理か、またはサーバが依頼とは同期せずに処理するメッセージ・タイプが通達型の非同期処理か。
2) サーバ上にデータが集中保管されているか、または分散保管されているか。
3) 複数のサーバにより処理される時、サーバ間の処理方式が同期処理であるか、または非同期処理であるか。
4) クライアントとサーバ間の処理方式が同期処理の時、依頼のメッセージ・タイプがトランザクション型であるか、または SQL 等の依頼応答型であるか。

以上の分類結果を一覧表にすると、分散処理システムのアーキテクチャ・スタイルは、表 5-1 に示す９つのスタイルになる[44]。この分類は、通常の C/S モデルに対してデータの分散形態と処理に時間の概念を導入したことになる。現実の大規模なシステムは、複数のスタイルの組み合わせで構成されている場合がある。

表 5-1　アーキテクチャ・スタイル

C/S 間の処理方式 ＼ データ種類（注-1）		集中	分散	
			同期処理	非同期処理
同期処理	トランザクション型	集中トランザクション スタイル	分散トランザクション スタイル	非同期トランザクション スタイル
	依頼応答型	集中 依頼応答 スタイル	分散 依頼応答 スタイル	非同期 依頼応答 スタイル
非同期処理	通達型	集中通達 スタイル	分散通達 スタイル	非同期通達 スタイル

（注-1）：サーバ間の処理方式（注-2）：メッセージ・タイプ

本章で使用する用語の意味を次に示し、表 5-2 に使用する記号を示す。

・「**データ種類**」は、使用するデータの保管形態を示している。データが一カ所に保管されている集中とデータが複数箇所に保管されている分散がある。但し、クライアント上の個人用ファイル及び個人用データベースは対象外とする。

・「**処理方式**」は、クライアント／サーバ間及びサーバ間の処理形態を示している。同期処理と非同期処理がある。

・「**トランザクション型**」とは、ACID 特性[22]を持ったトランザクション処理の形態である。ここで、ACID とは、原始性(Atomicity)、一貫性(Consistency)、独立性(Isolation)、耐久性(Durability)の略である。

・「**依頼応答型**」とは、クライアント側の依頼に対しサーバ側で同期的に処理がなされ、応答が返される形態である。

・「**通達型**」とは、クライアント側からの通達（メッセージ等）に対しサーバ側間で非同期に処理がなされる形態である。

表 5-2 本章で使用する記号

記号	意味
P	プレゼンテーション
AL_n	アプリケーション・ロジック
DM_n	データ管理
D_n	データ
⟷	トランザクション型メッセージ
⟷	依頼応答型メッセージ
⟷	通達型メッセージ

5.3.2 集中トランザクション・スタイル

単一のサーバ上に単一のデータベースがあり、クライアントとサーバ間の処理方式がトランザクション型の同期処理である。典型的な集中トランザクション・スタイルを図 5-4 に示す。

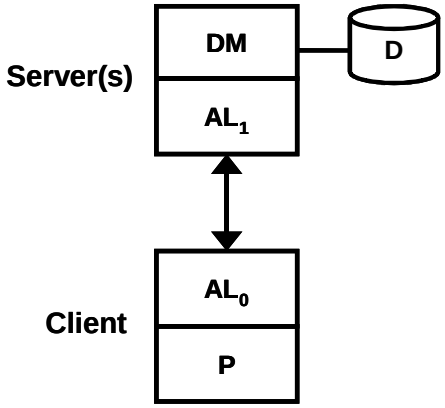


図 5-4　集中トランザクション・スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL_0 よりトランザクション型メッセージが AL_1 に送られる。AL_1 の処理結果で D の中のデータが DM により検索・更新・追加・削除される。そして処理結果がクライアントに返される。

5.3.3 分散トランザクション・スタイル

複数のサーバ上に複数のデータベースがあり、クライアントとサーバ間の処理方式がトランザクション型の同期処理であり、サーバ間の処理方式が同期処理である。典型的な分散トランザクション・スタイルを図 5-5 に示す。

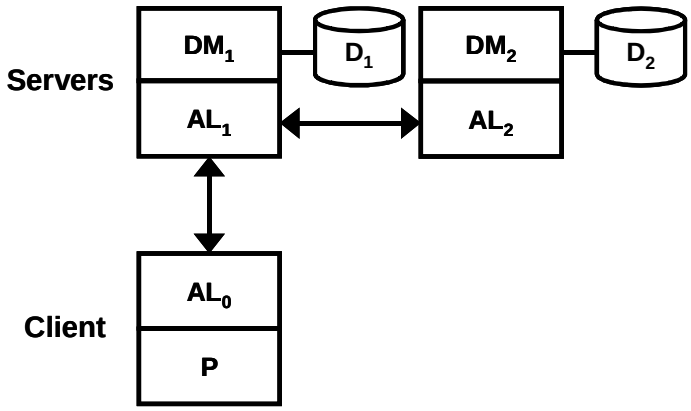


図 5-5　分散トランザクション・スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL_0 より、トランザクション型メッセージが AL_1 に送られる。そのメッセージの一部の処理結果で D_1 の中のデータが DM_1 により検索・更新・追加・削除される。そして処理されなかった又は処理された結果のトランザクション型メッセージが AL_2 に送られ、処理結果で D_2 の中のデータが DM_2 により検索・更新・追加・削除される。そして処理結果がクライアントに返される。必要に応じて、D_1 と D_2 の整合性を保つためのメカニズムが使用される。

5.3.4 非同期トランザクション・スタイル

複数のサーバ上に複数のデータベースがあり、クライアントとサーバ間の処理方式がトランザクション型の同期処理であり、サーバ間の処理方式が非同期処理である。典型的な非同期トランザクション・スタイルを図 5-6 に示す。

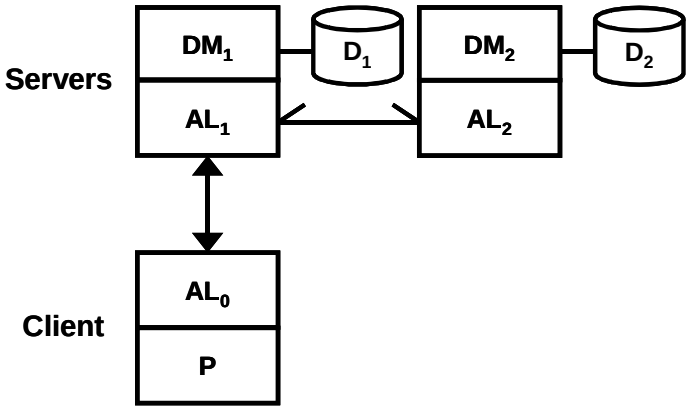


図 5-6 非同期トランザクション・スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL_0 より、トランザクション型メッセージが AL_1 に送られる。AL_1 の処理結果により D_1 の中のデータが DM_1 により検索・更新・追加・削除される。そして AL_1 の処理で作成された通達型メッセージが AP_2 に送られる。そして処理結果が、AL_0 のクライアントに返される。AL_1 からの起動要求等、適当なタイミングで、別のサーバ上の AL_2 より、既に送られている AL_1 よりのメッセージに従って、D_2 の中のデータが DM_2 により検索・更新・追加・削除される。

典型的な利用方法としては、適当なタイミングで、サーバ群を統合するサーバ上の D_2 の中のデータの一部（全部）を使用して、統合されるサーバ上の D_1 の中のデータを更新（ダウンロード）する。又は、その逆に D_1 の中のデータを使用して D_2 の中のデータを更新（アップロード）するトランザクション処理型のアプリケーションである。

5.3.5 集中依頼応答スタイル

単一サーバ上に単一データベースがあり、クライアントとサーバ間の処理方式が依頼応答型の同期処理である。典型的な集中依頼応答スタイルを図 5-7 に示す。

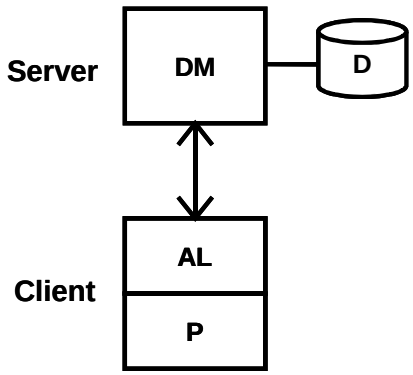


図 5-7　集中依頼応答スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL より依頼応答型メッセージ（SQL 形式のメッセージ等）が DM に送られる。メッセージ形式で表された要求により D の中のデータが DM により検索・更新・追加・削除される。そして処理結果がクライアントに返される。

5.3.6 分散依頼応答スタイル

複数サーバ上に複数のデータベースがあり、クライアントとサーバ間の処理方式が依頼応答型の同期処理であり、サーバ間の処理が同期処理である。典型的な分散依頼応答スタイルを図 5-8 に示す。

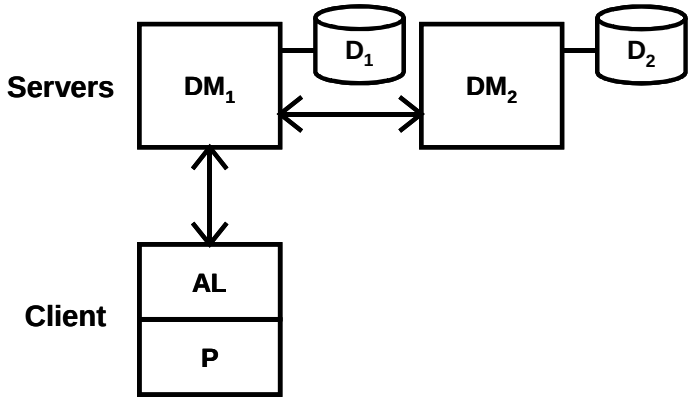


図 5-8　分散依頼応答スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL より、依頼応答型メッセージ（SQL 形式のメッセージ等）が DM_1 に送られる。そのメッセージの（一部の）要求により D_1 の中のデータが DM_1 により検索・更新・追加・削除される。そして処理されなかったメッセージが、または処理された結果のメッセージが DM_2 に送られ、D_2 の中のデータが DM_2 により検索・更新・追加・削除される。そして処理結果がクライアントに返される。必要に応じて、D_1 と D_2 の一貫性を保つためのメカニズムが使用される。

5.3.7 非同期依頼応答スタイル

複数のサーバ上に複数のデータベースがあり、クライアントとサーバ間の処理方式が依頼応答型の同期処理であり、サーバ間の処理が非同期処理である。典型的な非同期依頼応答スタイルを図 5-9 に示す。

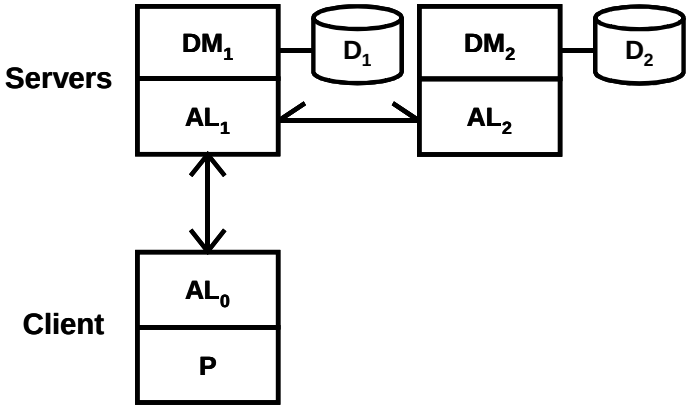


図 5-9　非同期依頼応答スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL_0 より、依頼応答型メ

ッセージ（SQL 形式のメッセージ等）が AL_1 に送られる。そのメッセージの要求により D_1 の中のデータが DM_1 により検索・更新・追加・削除される。そして AL_1 の処理で作成された新しい通達型メッセージが AL_2 に送られる。そして処理結果が、AL_0 のクライアントに返される。AL_1 からの起動要求等、適当なタイミングで、別のサーバ上の AL_2 より、既に送られている AL_1 よりのメッセージに従って、D_2 の中のデータが DM_2 により検索・更新・追加・削除される。

典型的な利用方法としては、適当なタイミングで、サーバを統合するサーバ上の D_2 の中のデータの一部（全部）を使用して、統合されるサーバ上の D_1 の中のデータを更新（ダウンロード）する、又は、その逆 D_1 の中のデータを使用して D_2 の中のデータを更新（アップロード）する依頼応答処理型のアプリケーションである。

5.3.8 集中通達スタイル

単一サーバ上に単一データベースがあり、クライアントとサーバ間の処理方式が非同期処理である。典型的な集中通達スタイルを図 5-10 に示す。

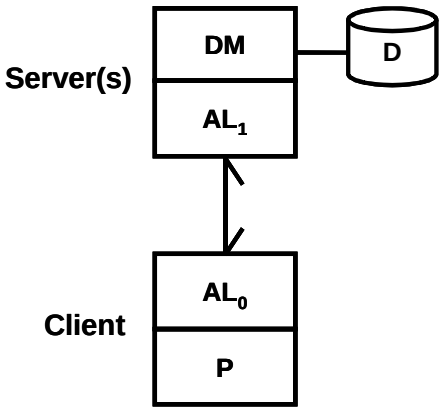


図 5-10　集中通達スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL_0 より、通知型メッセージ（文章等）が AL_1 に送られる。そのメッセージにより非同期に AL_1 により D の中のデータが DM により検索・更新・追加・削除される。

5.3.9 分散通達スタイル

複数のサーバ上に複数のデータベースがあり、クライアントとサーバ間の処理方式が非同期処理であり、サーバ間の処理が同期処理である。典型的な分散通達スタイルを図 5-11 に示す。

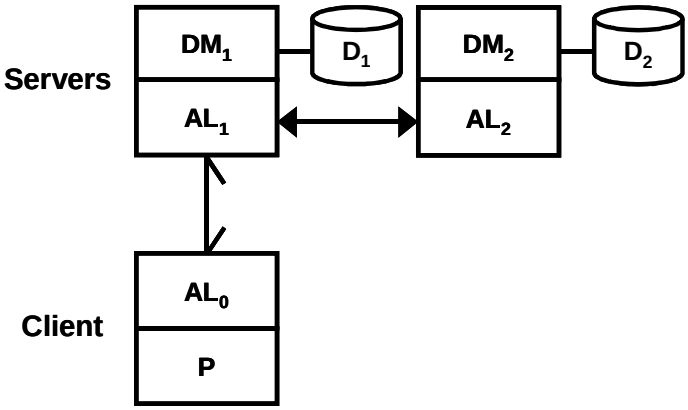


図 5-11　分散通達スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL_0 より、通達型メッセージ（文書等）が AL_1 に送られる。そのメッセージによりクライアントと非同期に AL_1 により D_1 の中のデータが DM_1 により検索・更新・追加・削除される。そして AL_1 の処理で作成された新しいトランザクション型（又は、依頼応答型）メッセージが AL_2 に送られる。AL_2 により D_2 のデータが DM_2 により検索・更新・追加・削除される。そして結果が AL1 に返される。

5.3.10 非同期通達スタイル

複数のサーバ上に複数のデータベースがあり、クライアントとサーバ間の処理方式が非同期処理であり、サーバ間の処理方式も非同期処理である。典型的な非同期通達スタイルを図 5-12 に示す。

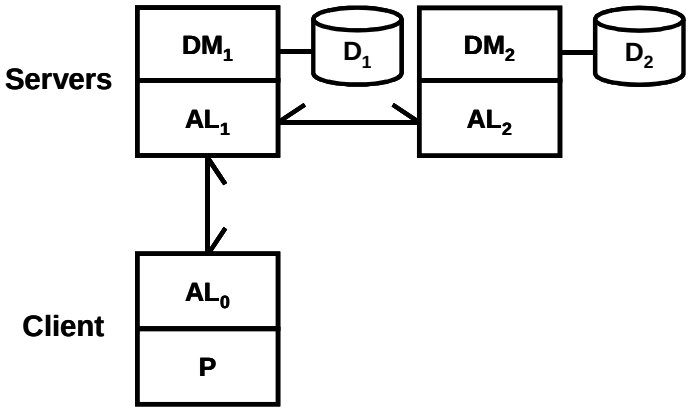


図 5-12　非同期通達スタイル

メッセージの流れと処理の流れを説明する。クライアント上の AL_0 より、通達型メッセ

ージ（文書等）が AL_1 に送られる。そのメッセージにより非同期に AL_1 により D_1 の中のデータが DM_1 により検索・更新・追加・削除される。そして AL_1 の処理で作成された新しい通達型メッセージが AL_2 に送られる。そのメッセージにより非同期に AL_2 により D_2 の中のデータが DM_2 により検索・更新・追加・削除される。

5.4 アーキテクチャ・スタイルの選択

分散コンピューティング環境下でアプリケーション・システムを開発するとき、アプリケーション・システムのアーキテクチャ・スタイルは重要な役割を演ずる。しかし、同じ課題を解決するシステムであっても、利用者、開発者そして管理者の要件や予算や開発期間の制約によって様々なアーキテクチャ・スタイルを使用することになる。アーキテクトが直感的に過去の積み重ねと経験から適切なアーキテクチャ・スタイルを選択しているのが現状である。コンサルタントは、お客様より新しいシステムの構築が要請されたときコンサルティング手法を使ってアーキテクチャ・スタイルを選択している。

本章では、経験の少ないアーキテクトが、分散処理システム環境下でビジネス・アプリケーションの適切なアーキテクチャ・スタイルの選択を支援する単純で実践的な手法を提案する[46,47,48]。この手法は、ABC の第一ステップ(1)の品質要件をベースに、適切なアーキテクチャ・スタイルの選択を支援することになる。品質特性は、特性軸として表現される。アーキテクトは、技術的環境下の適切なアーキテクチャ・スタイルを参照しながらアプリケーション・システムのアーキテクチャを設計する。そして開発者は、アーキテクチャ・スタイルに関連した評価済みのソフトウェア・ツールを使用してアーキテクチャをベースにしてアプリケーション・システムを実現する。このソフトウェア・ツールを OSFW ではプロダクトセットと称している。

5.4.1 アーキテクチャ・スタイルの特性化

アーキテクチャ・スタイルは、開発するアプリケーション・システムの構成要素の処理構造や処理スタイルを表現している。アプリケーション・システムのフレームワークを定義するアーキテクチャ・スタイルは、システムを利用したり開発したり管理したりする各種の参画者により選択され、システムの品質、開発保守管理コスト、そして開発期間に大きな影響を与える。アーキテクチャに関する要件は、機能に関する要件のように多くなく、最大約 20 要件[5]と言われているが、本章では、幅広い視点でなく七つの視点に制限した。これは、より少ない視点が実践的な適用性があると考えているからである。

まず、ソフトウェア品質を論じる。JIS では、ソフトウェアの品質特性として、機能性、信頼性、使用性、効率性、保守性及び移植性が定義されている[21,25]。これらの品質特性

は、21 の品質副特性に細分化されている。機能性は、合目的性、正確性、相互運用性、標準適合性とセキュリティに細分化される。信頼性は、成熟性、障害許容性と回復性に細分化される。使用性は、理解性、習得性と運用性に細分化される。効率性は、資源効率性と時間効率性に、そして保守性は、解析性、変更性、安定性と試験性に細分化される。移植性は、環境適応性、設置性、規格適合性と置換性に細分化される。これらの特性は、ソフトウェアのライフサイクル全般で品質要件の定義や評価のために使われる。

アーキテクチャ・スタイルの特性をシステムのライフサイクル全般に参画する者の視点から考察する。分散処理型のアプリケーション・システムを実現するとき、システム・アーキテクトは利用者や開発者そして管理者の要件をベースにして鳥瞰的な視点でアーキテクチャ・スタイルを選択し、そのスタイルを参照しながらアプリケーション・システムを設計する。

システムの利用者の視点から、要求する機能の機能性、信頼性、使用性、効率性と他環境への移植性が品質特性の考慮の対象になる。しかし、アーキテクチャ・スタイルの選択段階では、セキュリティ副特性以外の機能性は前提条件であり、信頼性と使用性及び移植性は詳細設計段階の特性である。従って、利用者のセキュリティと応答性の重要性より「データ・セキュリティ」と「ユーザ応答性」が特性になる。

システムの開発者の視点からは、ソフトウェアの全ての品質特性が考慮の対象になる。しかし、アーキテクチャ・スタイルの選択段階では、機能性は開発の前提条件であり、信頼性、使用性、保守性と移植性は詳細設計段階の特性である。効率性は、考慮の対象である。従って、システム構造と処理方式の要件より、「C/S 領域」、「サーバ独立性」と「データ分散性」が特性になる。

システムの管理者は、限られた人的資源と時間枠の中でソフトウェアの品質を最適にする。従って、システム開発を管理する要件より「システム予算」と「システム提供」が特性になる。

上記の考察をベースにして、次の七つの特性軸を設定した。アーキテクチャ・スタイルの特徴を可視化するために、各特性軸の特性値毎に付加されている括弧内の数値を使用し、レーダチャート形式で特性図を表現する。付加している数値は、高機能で、信頼性があり、効率の良い、使い勝手の良いシステムを安価に開発したいというソフトウェア品質及びシステム開発の一般的な要件をベースにしている。経験の少ないアーキテクトがこのスタイルを使用するために、出来る限り“ ハイ／不明／いいえ ”のような単純で理解が容易な特性値を設定している。数値が大きい特性値の方が、機能性と効率性が高いと判断している。

1) データ・セキュリティ：

データ・セキュリティは、一定の予算と期間内にシステムを開発するときの利用者の視点からの特性軸である。特性値は、利用者が安全な処理を要請する「高い(3)」と、それなりの処理を要請する「低い(1)」と、利用者がセキュリティ・レベルの決定を必要としない時やセキュリティの情報が無い時は「どちらでも(2)」に分類する。

アーキテクチャ・スタイルは、トランザクション型、依頼応答型、通達型に関わらず、データを集中した時の方がセキュリティ対策を採り易いので、セキュリティは「高い」である。データを分散したときは、セキュリティは「低い」である。データを分散しても集中しても良いときや、データを分散する要件が無いときは、「どちらでも」である。

2) ユーザ応答性：

クライアント（ユーザ）がサーバに処理を依頼するときの応答性の特性軸である。クライアントがサーバに処理を要求するのみで応答を要求しない「通達(1)」と、処理要求に対してその結果報告を要求する「応答(3)」と、どちらでも良い「どちらでも(2)」」に分類する。

トランザクション型と依頼応答型のアーキテクチャ・スタイルは、「応答型」と特徴付けられる。通達型は「通達」と特徴付けられる。

3) C/S 領域：

クライアント（利用者）とサーバ間及びサーバ間の所属する領域（ネットワークの形態）に関する特性軸である。この特性値は、クライアントとサーバ(C/S)間が近距離で LAN のみでサーバと接続する「LAN(3)」と、C/S が広域に属し WAN で接続している「WAN(1)」と、そして、C/S 間の距離が一様でなく、ネットワークに LAN と WAN が混合している「LAN/WAN(2)」に分類する。

WAN 型のネットワークは、C/S 間の処理方式としてトランザクション型を一般的に使用する。LAN 型は、通信速度や通信量に対して WAN 型と比較して制約が少ないため、C/S 間の処理方式としてトランザクション型でも、依頼応答型でも、通達型でも使用可能である。依頼応答型のスタイルでは、C/S 間のネットワーク間の通信量が多いため一般的に LAN 型のネットワークが使われる。

4) サーバ独立性：

複数のサーバで依頼された処理を遂行するときのサーバ間での処理方式の特性軸である。この特性値は、サーバ上の処理が要求者（サーバ）と非同期に実行される「独立(3)」と、サーバ上の処理が要求者と同期して実行される「依存(1)」と、そしてサーバ間の関係を決

定できない「どちらでも(2)」に分類する。
　非同期型のスタイルの特性値は、「独立」である。他のアーキテクチャ・スタイルの特性値は、「依存」である。

5)　データ分散性：
　取り扱うデータの保管に関する特性軸である。この特性値は、分散保管されローカル処理も可能な「分散(3)」と、一個所に集中できるデータ量であり、業務的に一個所の方が処理し易い「集中(1)」と、そして集中保管にも分散保管にも適合しどちらとも判断できない「どちらでも(2)」に分類する。
　集中型のスタイルの特性値は、「集中」である。分散型のアーキテクチャ・スタイルの特性値は、「分散」である。

6) システム予算：
　システムを開発し管理するための予算（コスト）の特性軸である。この特性値は、少ない予算の「手ごろな(3)」と、充分な予算の「充分(1)」と、そして普通の予算であったり、予算情報がないときの「どちらでも(2)」に分類する。
　アーキテクチャ･スタイルの特性値として､我々の経験より習得した次の規則を適用する。トランザクション型のアプリケーション・システムの開発には、比較的余分なコストが必要であり、依頼応答型の開発は比較的低コストである。通達型の開発は通常のコストが必要である。

7) システム提供：
　システムの提供時期に関する特性軸である。この特性値は、システムを短期間に提供することが要請されている「短い(3)」と、充分な開発期間が与えられている「充分(1)」と、そして決めることが出来ないの「どちらでも(2)」に分類する。
　アーキテクチャ･スタイルの特性値として､我々の経験より習得した次の規則を適用する。トランザクション型や分散型スタイルのアプリケーション・システムの開発期間は、比較的長めである。依頼応答型と通達型の開発期間は、比較的短めである。

　前章で設定した各アーキテクチャ･スタイルに各特性軸の上記の基準を適用した結果を一覧表として表 5-3 に示す。各アーキテクチャ・スタイルの特性値を視覚化した特性図を図 5-13 に示す。

表 5-3　アーキテクチャ・スタイルの特性値一覧

特性軸 / アーキテクチャ・スタイル	データ セキュリティ	ユーザ応答性	C/S領域	サーバ独立性	データ分散性	システム予算	システム提供
集中トランザクション・スタイル	高い(3)	応答(3)	WAN(1)	依存(1)	集中(1)	充分(1)	長い(1)
分散トランザクション・スタイル	低い(1)	応答(3)	WAN(1)	依存(1)	分散(3)	充分(1)	長い(1)
非同期トランザクション・スタイル	低い(1)	応答(3)	LAN/WAN(2)	独立(3)	分散(3)	充分(1)	長い(1)
集中依頼応答スタイル	高い(3)	応答(3)	LAN(3)	依存(1)	集中(1)	手ごろ(3)	短い(3)
分散依頼応答スタイル	低い(1)	応答(3)	LAN(3)	依存(1)	分散(3)	手ごろ(3)	どちらでも(2)
非同期依頼応答スタイル	低い(1)	応答(3)	LAN/WAN(2)	独立(3)	分散(3)	手ごろ(3)	短い(3)
集中通達スタイル	高い(3)	通達(1)	LAN/WAN(2)	依存(1)	集中(1)	どちらでも(2)	短い(3)
分散通達スタイル	低い(1)	通達(1)	LAN/WAN(2)	依存(1)	分散(3)	どちらでも(2)	どちらでも(2)
非同期通達スタイル	低い(1)	通達(1)	LAN/WAN(2)	独立(3)	分散(3)	どちらでも(2)	短い(3)

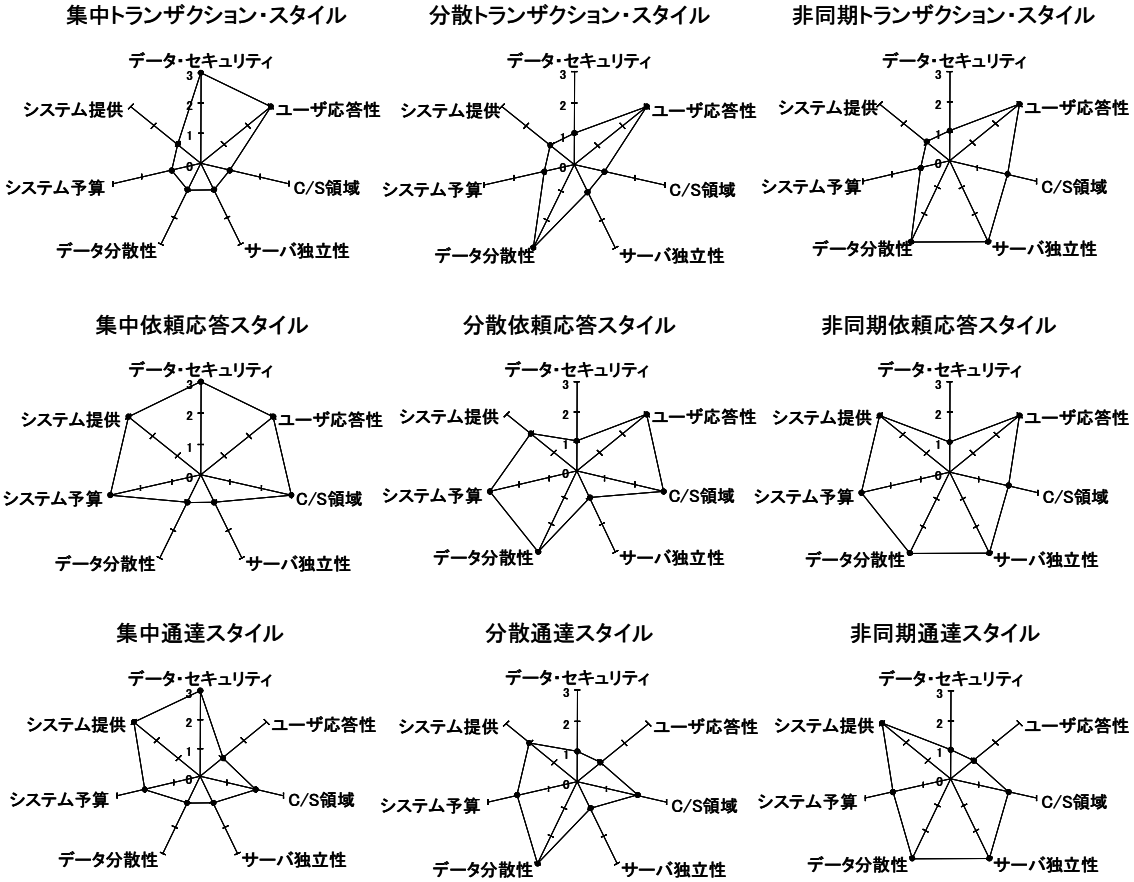


図 5-13　アーキテクチャ・スタイルの特性図

5.4.2 アーキテクチャ・スタイルの選択手法

アーキテクチャ・スタイルを選択するとき、システム・アーキテクトは、利用者や開発者そして管理者の要件を鳥瞰的な視点から考察する。通常、全ての要件は満たされない。アーキテクチャ・スタイルの選定は、実現手法、アプリケーション・システムの処理構造と処理形態、システムを実現し管理するためのソフトウェア・ツールを選択するための重要な決定である。

本節で、アプリケーション・システムに適合するアーキテクチャ・スタイルを選択する手法を説明する。そのために、前節で導入したアーキテクチャ・スタイルの特性軸と特性値を使用する。アプリケーション・システムにどのアーキテクチャ・スタイルが適合するかを計測するために、「距離」と「サイズ」の尺度を導入し、計測と選択手法を示す。そしてまた、レーダーチャート間の類似性が選択手法として有用であることを示す。

5.4.2.1 計測尺度

距離とサイズの尺度の定義を次に与える。

1) 距離の尺度

アプリケーション・システム(X)と９つの各アーキテクチャ・スタイル(S_j)の７つの特性軸毎に特性値間の差異を計測する。特性軸毎の特性値の差異の絶対値の合計をアーキテクチャ・スタイルとの距離とする。

■ X と i 番目のアーキテクチャ・スタイル間の距離 ＝ $\sum_{j=1}^{7} | S_{ij} - X_j |$

但し、Sij = i 番目のアーキテクチャ・スタイルの j 番目の特性値

Xj = Xの j 番目の特性値

2) サイズの尺度

各特性値の合計をアーキテクチャ・スタイルのサイズとする。

■ i 番目のアーキテクチャ・スタイルのサイズ ＝ $\sum_{j=1}^{7} S_{ij}$

但し、Sij = i 番目のアーキテクチャ・スタイルの j 番目の特性値

特性値として使用する数値は、数値の大きいスタイルの方を機能性や効率性が高い、また短期開発で低開発コストであると見なしている。従って、サイズが大きいスタイルが、一般的なソフトウェア品質やシステム開発の要件により合致することになり、複数のアーキテクチャ・スタイルが選択されることもある。この選択手法は、アプリケーション・システムの特性図とアーキテクチャ・スタイルの特性図間の類似性を調べることとも同一に

なる。一般的に、距離が近いとき、特性図の類似度は高くなる。

5.4.2.2 アーキテクチャ・スタイルの選択手順

アーキテクチャ・スタイルを選択する手順は、次の通りである。

1) アプリケーション・システムの各特性軸の特性値を決定する。

2) アプリケーション・システムと各アーキテクチャ・スタイル間の距離を計測する。この時、手順 1)のアプリケーション・システムの特性値と表 5-3 のアーキテクチャ・スタイルの特性値を使用する。

3) 距離が最小であるアーキテクチャ・スタイルがシステムのアーキテクチャ・スタイル候補である。

4) アプリケーション・システムとアーキテクチャ・スタイル間の距離が同じとき、アーキテクチャ・スタイルのサイズが最大のものがシステムのアーキテクチャ・スタイル候補である。

5) アーキテクチャ・スタイルのサイズが同じ時、特性図がより類似するアーキテクチャ・スタイルが候補である。

この選択処方を適用した後、アプリケーション・システムは、アプリケーション・システムの要件と各アーキテクチャ・スタイル間の距離とサイズと類似度を使って、アーキテクチャ・スタイルが適応順に並べられる。

アプリケーション・システムに複数のクライアントが存在する場合、各クライアント毎にサブシステムに分割することが出来る。そして、各クライアントの要件をベースにアーキテクチャ・スタイルを選択するために、上記の５つの手順を適用する。

5.5 実アプリケーション・システム適用試行事例

第 5.3 節で提案したアーキテクチャ・スタイルの適合性を検証するために、実稼働中のアプリケーション・システムへ適用した。本節では、実施した実アプリケーション・システムの代表的なものを四つ記述する。

5.5.1 P 株式会社 チケット予約システム

このシステムは、全国 620 店舗に展開されている WAN 型の専用回線で接続された 800 台の専用端末から映画やコンサート等のチケットを会員に予約販売する。チケットは、土曜日と日曜日の朝一番に全国販売拠点で一斉に販売され、購入希望会員には即時に予約結果が知らせる。チケット販売では、各販売拠点より、大阪地域、北海道地域、名古屋地域、そして東京地域におかれた各地域サーバ経由で本社に設置されている本社サーバのチケットの予約・販売状況が検索されている。また会員情報の管理や検索、チケット代金の支払い・精算業務のシステムでもある。このシステムの開発では、システムの拡張性とセキュリティ度に関して、特段の要求がなされていなかった。システムの性格上、予約データのリカバリー機能が要求され、ある程度の開発コストや運用コストの増加は容認されていた。

このシステムの要件を表現した評価値を表 5-4 に示す。このシステムのアーキテクチャ・スタイルの候補は、表 5-5 に示すように距離が最小である集中トランザクション・スタイルと分散トランザクション・スタイルである。さらに、このシステムの特性図を図 5-14 に示す。この特性図は、集中トランザクション・スタイルより分散トランザクション・システムの特性図に類似している。

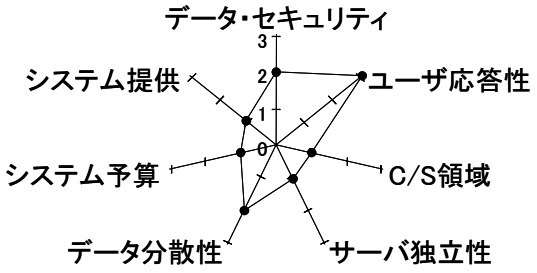


図 5-14　チケット予約システムの特性図

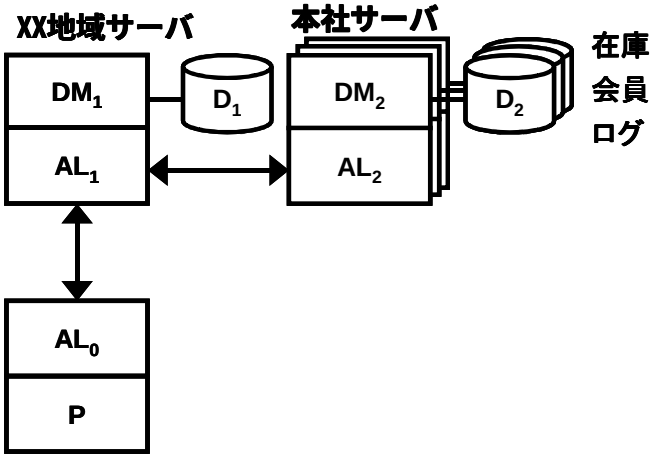


図 5-15　チケット予約システム

図 5-15 に本社サーバの負荷を分散するためにローカル・サーバを利用した分散トランザクション・スタイルを適用したチケット予約システムを示す。

表 5-4　事例の特性値一覧

特性軸 / 被評価システム	データセキュリティ	ユーザ応答性	C/S領域	サーバ独立性	データ分散性	システム予算	システム提供
チケット予約	どちらでも(2)	応答(3)	WAN(1)	依存(1)	どちらでも(2)	充分(1)	長い(1)
会計エントリー業務	高い(3)	応答(3)	LAN/WAN(2)	独立(3)	分散(3)	どちらでも(2)	どちらでも(2)
フィールド・エンジニア支援	低い(1)	通達(1)	LAN/WAN(2)	依存(1)	分散(3)	手ごろ(3)	短い(3)
お客様申し込み工事支援	低い(1)	通達(1)	LAN/WAN(2)	独立(3)	分散(3)	どちらでも(2)	短い(3)

表 5-5　アーキテクチャ・スタイル間の距離一覧

アプリケーション・システム / アーキテクチャ・スタイル	チケット予約	会計エントリー業務	フィールド・エンジニア支援	お客様申し込み工事支援
集中トランザクション・スタイル	2	7	11	12
分散トランザクション・スタイル	2	7	7	8
非同期トランザクション・スタイル	5	4	8	5
集中依頼応答スタイル	8	7	7	10
分散依頼応答スタイル	7	6	4	7
非同期依頼応答スタイル	9	4	4	3
集中通達スタイル	8	7	5	6
分散通達スタイル	7	6	2	3
非同期通達スタイル	10	5	3	0

5.5.2　K 製薬株式会社会計エントリ業務システム

このシステムは、会計データを全国 10 個所に点在する事務所に於いて発生ベースで入力し、事務所毎に累積されたデータを本社のホスト・システムに定期的に集め、処理するアプリケーション・システムである。各事務所での入力処理は、入力データのチェック等処理の即時性が要求されている。会計データは事務所のサーバに累積され、定期的に本社サーバに集められて処理される。事務所内の機器は、LAN 接続されている。事務所と本社間は WAN 接続である。システムは、事務所の拡大等のため拡張性が要求されている。また会計データを取り扱うためセキュリティが求められている。開発と運用管理コストに関しては、特に言及されていなかった。

このシステムの要件を表現した評価値を表 5-4 に示す。このシステムのアーキテクチャ・スタイルの候補は、表 5-5 に示すように距離が最小である非同期トランザクション・スタイルと非同期依頼応答スタイルである。さらに、このシステムの特性図を図 5-16 に示す。この特性図は、非同期トランザクション・スタイルと非同期依頼応答スタイルの特性図に類似している。

図 5-17 で示す実稼動システムでは、非同期トランザクション・スタイルと非同期依頼応答スタイルのサイズが同一であるので、データ分散性、C/S 領域およびサーバ独立性の指標値が一致し、特性図がより類似する非同期トランザクション・スタイルを適用している。

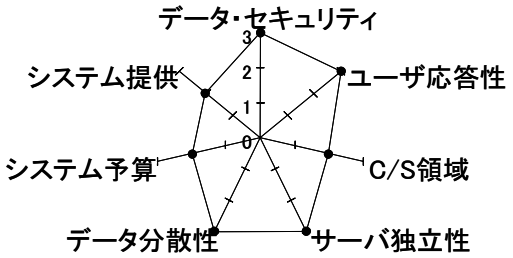


図 5-16　会計エントリ業務システムの特性図

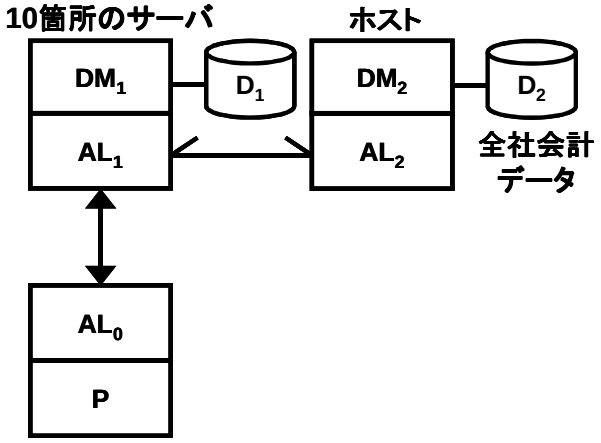


図 5-17　会計エントリ業務システム

5.5.3　C 株式会社フィールド・エンジニア支援システム

このシステムは、販売した商品の修理を担う FE（フィールド・エンジニア）の事務作業を支援する。修理依頼を受けた各地のフロント担当者は、受付データを入力し、サーバから FE のハンディ端末に訪問先を指示する。FE は、修理指示データをもとに現場に直行し作業を行い、修理結果を現場でハンディ端末に入力し作業完了書も出力する。作業完了データは、当日あるいは翌日にフロントサーバに送信される。フロントサーバに送信されたデータは、東京本社本部サーバに蓄積される。このシステムは、関連するデータが分散している。ハンディ端末とフロントサーバ間の処理は同期していないが、フロントサーバと本部サーバ間のデータは同期する必要がある。ネットワークは LAN と WAN が混合している。システムの拡張性に関しては何も言及されていない。またセキュリティに関しても強く求められていない。しかし、リカバリーの確保に関しては強く求められていた。

このシステムの特性図を図 5-18 に示す。この特性図は、分散通達スタイルの特性図に類似している。

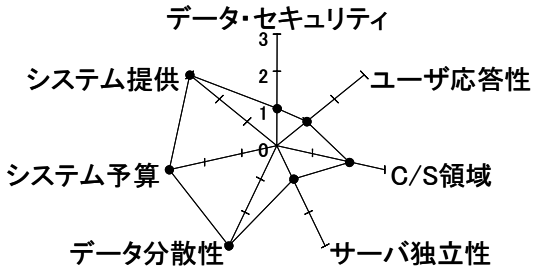


図 5-18　FE 支援システムの特性図

図 5-19 で示す FE 支援システムでは、分散通達スタイルを適用している。

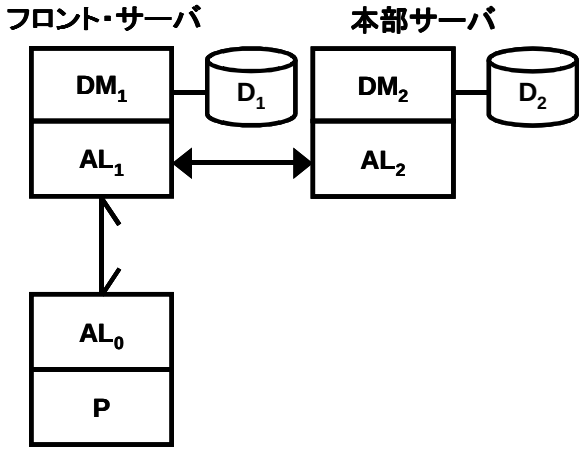


図 5-19　フィールド・エンジニア支援システム

5.5.4　C 電力株式会社お客様申込み工事支援システム

このシステムは、お客様よりの工事の申込みを支援する。お客さまからの工事申込みの受付データはホスト・コンピュータの申込み内容データベースに蓄積され、一定時間ごとに営業所サーバへ自動的にダウンロードされる。営業所では、サーバ上のデータベースを使用して、当日分の施工予定データを区域別に自動的に振り分けて、各サービス・エンジニア分（携帯端末機器用）のデータを作成する。このデータは各携帯端末機へダウンロードされ、サービス・エンジニアはそのデータを基に施行する。工事終了後、サービス・エンジニアは施工結果を携帯端末機に入力する。帰社後サービス・エンジニアは、携帯端末に入力された施工結果を営業所のサーバにアップロードする。データの正当性が確認された工事竣工情報は、ホスト・コンピュータへ取り込まれる。お客様申込み工事支援システ

ムは、受付業務と工事支援業務より構成されている。工事支援業務では、必要なデータは分散している。工事支援では移動端末が使用されるが、応答性は求められず、営業所サーバと本社ホスト間も非同期に作業がなされている。ネットワークに関しては、LAN と WAN が混在している。システムの拡張性は機能追加を含め求められている。セキュリティに関しては求められていず、コストに関しても特に言及されていなかった。

このシステムの特性図を図 5-20 に示す。この特性図は、非同期通達スタイルの特性図に一致している。

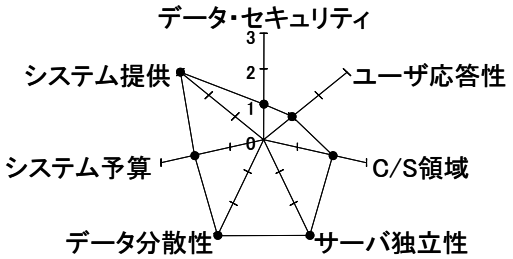


図 5-20　お客様申込み工事支援システムの特性図

図 5-21 で示すお客様申し込み工事支援システムでは、受付業務は集中トランザクション・スタイル、そして工事支援システムは非同期通達スタイルが適用している。

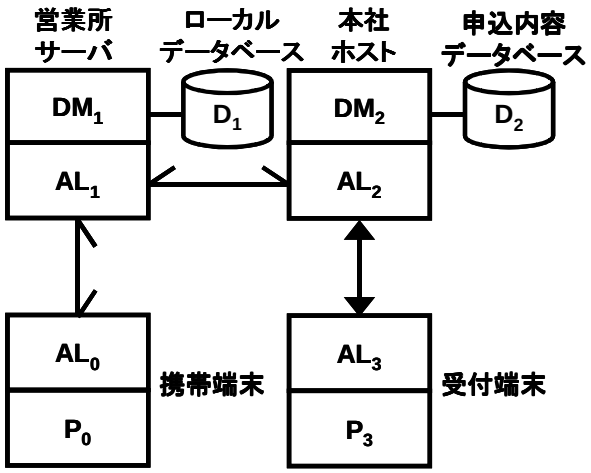


図 5-21　お客様申込み工事支援システム

5.5.5　評価

各事例において、七つの特性軸の特性値を決めることは、特性軸間の関連性があったとしても、“ハイ／不明／いいえ”のような単純な値を使用しているので比較的簡単な作業である。単純な問い合わせシステムを使うことによって、システム・アーキテクトは、全て

の特性軸の値を簡単に決めることが出来る。特性軸に関連した要件を理解していれば、“ハイ”か“いいえ”の値を使用し、関連する要件に関する情報が存在しない時や要件が理解できない時は“不明”の値を使用することになる。

アーキテクチャ・スタイル間の距離は、別の見方をすると優先順位の概念の導入になる。選択の優先順位として使用すれば、次候補の選択が容易になる。選択に於いて次候補が見えていることは重要である。

本章では、特性軸間にはその重要性に関して差がないものと仮定して距離とサイズを計測している。しかし、機能を制限しても、低い開発コストでセキュリティを保持したシステムを開発しなければならない場合がある。そのような強い要請に対しては、特性軸毎に重み付けを加えて距離とサイズを計測する方法がある。この場合、企業毎やシステム環境毎に何種類かの重み付け計数を作ることが出来る。しかしながら、本章の事例で示したように多くの場合、特性軸に重み付けを加えることなく実践的に使用することが出来る。

5.6 関連する研究

分散処理システムのアーキテクチャ・スタイルは、コンピュータ・ハードウェアの多階層構成とシステム機能を論理的に分割した協調型の C/S 処理の二つに分けることができる。前者は、エンタープライズ・サーバ（メインフレームやホスト）、サーバ、ワークステーションとコンピュータ・ハードウェアを三階層に分離した形態がその例である。後者は、クライアントからの要求に対してサーバが応答する C/S 処理方式がその例である。後者の例として、Berson[6]は、分散処理システムを分散プレゼンテーション、リモート･プレゼンテーション、分散ビジネス・ロジック、リモート・データ管理、分散データ管理の５つのモデルに分類している。同様にガートナグループ[8]も Berson のモデルに類似した５つのモデルを定義し、分散処理システムの参照モデルとして使用している。これらのモデルは、プレゼンテーション機能、アプリケーション・ロジック機能、データ管理機能間での機能分散を基本にしているが、非同期処理が考慮されていないためグループウェア型のアプリケーションや電子メール型のアプリケーションをモデル化することが困難である。Berson の分類とモデルの名称が若干異なるがガートナグループのモデルを図 5-22 に示す。

IBM 社は、C/S システムの構築ガイド[16]の中で C/S システムを６つのテンプレートに分類している。それらは、フロント・エンディング、資源集中、ホスト分散ロジック、LAN 分散、データ・ステージングとマルチ・アプリケーションである。これらのテンプレートは、1990 年代の初期までに IBM の典型的なユーザによって設計されたり構築されたりした約 50 の実世界のソリューションを調査した結果である[79]。しかしながら、これらのテ

ンプレートをビジネス・アプリケーションに適用するときの、ユーザにとっての明確な選択基準が提示されていない。また、テンプレートは C/S システムの様々な処理形態をカバーしていない。

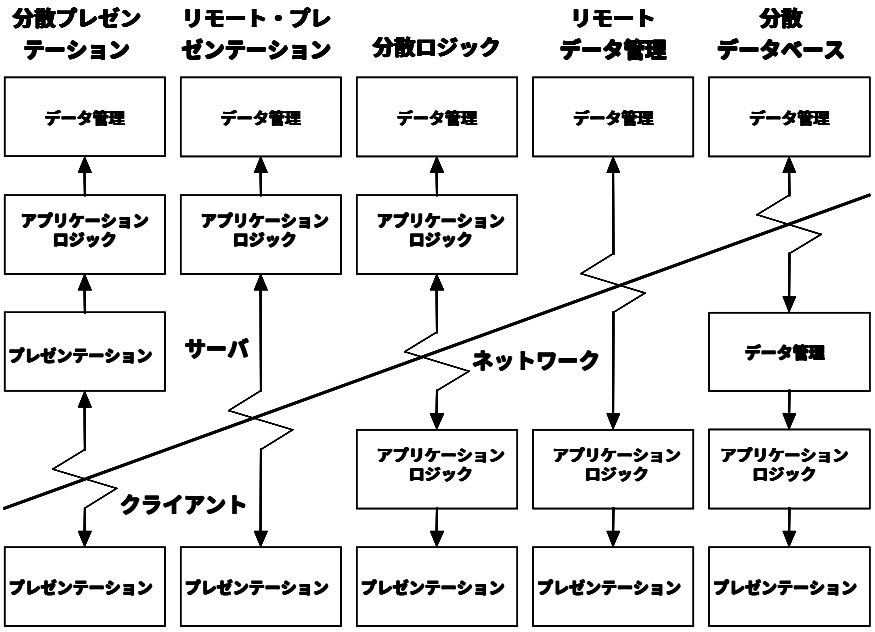


図 5-22　分散コンピューティング・モデル

森澤は、メインフレーム、Unix、PC が混在する 1990 年代中旬の情報処理社会で、メインフレームのみを経験しているユーザにも理解し易い直感的な７つのモデルを提案し[40]、モデルに対応した COTS プロダクトの組み合わせ[41]やアーキテクチャ設計への適用例[43]を示している。７つのモデルは、集中トランザクション処理、分散トランザクション処理、リモート・データ処理、分散データ処理、集中メッセージング処理、データ・ステージング処理、そして連鎖型メッセージング処理である。これらのモデルは、1990 年代中頃に出現し始めた多くの COTS プロダクトがカバーする領域を大別したものであり、基本的には、モデルを実現するオープンなプロダクトが市場に存在していることを前提にしている。従って、モデル発表以降に急速に発展したインターネット、エクストラネット、イントラネット、モバイル等の情報技術の実用化による新しい COTS プロダクトの登場に合わせて、モデルの見直しが必要になってきている。特に、モバイルとデータが垂直に分散された時の遅延処理の導入が必要である。

プラットフォームの多階層環境や協調型の C/S 処理のモデル化と異なって、過去の実際に稼動したソフトウェアをモデル化したアーキテクチャ・スタイルが発表されている。

Shaw はその著書[77]で、代表的な７つのアーキテクチャ・スタイルを紹介している。それらは、パイプとフィルター、データ抽象とオブジェクト指向構成、イベント駆動と暗黙駆動、レイヤー・システム、リポジトリ、インタプリタ、そしてプロセス制御である。これらのスタイルは、経験より得られたスタイルの集合である。Show は、アーキテクチャ・スタイルの機能ベースの分類も報告している[78]。それらは、データフロー(Data flow)スタイル、呼出復帰(Call-and-return)スタイル、相互作用(Interactive)スタイル、データ中心リポジトリ(Data-centered repository)、データ共有(Data sharing)スタイルと階層(Hierarchical)スタイルである。これらのスタイルは、ビジネス・アプリケーション・システムを構築するという視点から考察すると、アーキテクチャ・スタイルを選択するときの指針が明確にされていないため、参照モデルとして使用することが困難である。

ソフトウェア・アーキテクチャの分析手法として、シナリオをベースに点数付けする SAAM[30]と品質特性を類推するモデルを使用する ATAM[31]がある。SAAM は、システムの利用者と開発者の両者から期待する又は予想するシステムの簡潔な説明（シナリオ）を基に、ソフトウェア・アーキテクチャを分析する手法である。ATAM は、アーキテクチャを具体化したものが要件を満たしているか否かを明らかにするために、変更可能性、効率、信頼性、セキュリティ等の複数の品質属性を考慮したアーキテクチャ・レベルの設計を評価する手法である。この方法は、これら属性間のトレードオフ点を明確にし、利用者、開発者、顧客、保守者のような関係者間のコミュニケーション手段を提供している。ATAM は、リスク緩和手法、即ち複雑なソフトウェア・システムのアーキテクチャ内のリスクの可能性がある領域を指摘する手法であることを意味している。これらは、アーキテクチャ・スタイルの選択手法ではない。

アーキテクチャの選択に関しては、岸が意思決定の手法 AHP(Analytic Hierarchy Process = 階層的意思決定法) をアーキテクチャの選択方法としての適用性を報告している[32]。この方法はまだ試行の段階であり、経験の少ないアーキテクトには使用が困難である。

5.7 まとめと今後の課題

データの所在場所とクライアントとサーバ間およびサーバ同士間の処理方式に焦点を当てて、分散処理システムを九つのアーキテクチャ・スタイルに分類した。この分類は、1996 年に森澤[40]が発表した分散処理システムの処理モデルをその後の情報技術の進展に合わせて再検討した結果である。九つのアーキテクチャ・スタイルの適用例をまとめると次のようになる。

1）集中トランザクション・スタイルは、アプリケーション・システムの構築に市販のトランザクション管理プログラムを利用する単一データベースによる追加更新が主体のリアルタイム処理に適する。アプリケーションの例としては、受発注管理、商品在庫管理、株式注文処理、生産管理、小売 POS システム等がある。

2）分散トランザクション・スタイルは、複数のデータベースへの更新や問い合わせが複雑にからむ基幹系リアルタイム処理に適する。メッセージの流れを制御するために、市販のトランザクション管理プログラムのメッセージ・キューイング機能などを利用する非同期処理を伴う場合は、非同期処理を含んだスタイルに分類される。アプリケーションの例としては、銀行の勘定系システム、座席予約システム、工場生産管理システム等がある。

3）非同期トランザクション・スタイルは、データの非同期な共有による情報の管理と活用を促進する基幹系アプリケーション・システムに適する。アプリケーション・システムの例としては、支店や部門別に収集されたトランザクション・データの本社へのアップロード、部門サーバによって収集された受注データの本社サーバへのアップロードと本社サーバでの集中受注処理などである。

4）集中依頼応答スタイルは、意思決定支援システムに代表されるいわゆる EUC（エンドユーザ・コンピューティング)、および問い合わせ応答型の単純なリアルタイム処理に適する。EUC のアプリケーションの例には、予算計画、財務分析、市場調査・分析、売上分析、所要量計画、需要予測などの各種統計・分析・報告がある。問い合わせ応答処理のアプリケーションの例には、顧客サービス、営業支援、各種照会処理、情報提供サービスなどがある。

5）分散依頼応答スタイルは、複数のデータベースやファイルを同時にアクセスする EUC および問い合わせ中心の対話型処理に適する。このスタイルでは、すでに存在するデータベースの共有による情報の有効活用ができる。集中依頼応答スタイルのアプリケーションはほとんど含まれるが、他に、全社売上統計、全社生産性データ分析などがある。

6）非同期依頼応答スタイルは、データの非同期な共有による情報の管理と活用を促進するアプリケーションに適する。アプリケーションの例としては、業務系データベースの一部のダウンロードによる情報活用、例えば、大福帳や多次元データベースなどによる DSS、EIS などである。

7）集中通達スタイルは、グループ内あるいは組織内での単純なワークフローの自動化に適する。アプリケーションの例としては、電子メールによる伝票・社内文書の配送と回付およびイベント通知、内部文書のファイリング、業務フローの実行・監視・報告

などがある。

8）分散通達スタイルは、モバイル・クライアントからサーバ上のエージェント・アプリケーション（クライアントの代理機能）を用いた分散トランザクション処理や分散データ処理を行うのに適する。

9）非同期通達スタイルは、独立した複数のアプリケーションやシステム同士の連携による疎な統合を行うのに適する。アプリケーションの例としては、グループ間や組織間のワークフロー、企業内システム間のアプリケーション連携による統合、や EDI などによる企業間のシステム連携などがある。

第 5.2 節で提案したアーキテクチャ・スタイルを使用して、アプリケーション・システムを開発するときに、適切なアーキテクチャ・スタイルを選択する手法を提案した。提案した手法では、アーキテクチャ・スタイル間のように絶対的な優劣基準が存在しないときに、適切なアーキテクチャ・スタイルを選択するために、相対評価を採用した。まず、利用者の視点や開発者の視点及び管理者の視点よりアーキテクチャ・スタイルを特徴付ける特性軸とその特性値を導入した。アーキテクチャ・スタイルの適合性を計測するために「距離」と「サイズ」の計測尺度を導入し、アプリケーション・システムの要件を表現した特性図と各アーキテクチャ・スタイルの特性図間の類似性を使用する単純で実践的な選択手法を提案した。この手法を、本番稼働しているアプリケーション・システムを使用して、適合性の評価とこれらの有効性を確かめた。

本論文では言及していないが、分散処理システムでは、アプリケーション・ロジックをどのように分割し分散配置するかも重要な視点である。これらの課題については、今後の研究課題とした。

6. おわりに

6.1 まとめ

本論文の第2章で、1980年代の前半に活発に研究した論理型言語の事務処理問題への適用方法を論じた。この方法は、1980年代前半のProlog処理系の処理効率問題が方法論のさらなる研究を後押しすることなく、その後の論理型言語に対する実務家や研究者の関心の衰退ともに、事務処理分野では、ビジネスの視点より棚上げされ忘れられているアプリケーション・モデリング方法であった。しかし、近年の情報技術の発展により、Prolog処理系の効率問題が解消の方向にあることと、システム間で送受信されるデータの表現形式としてXMLが標準になろうとしている。入出力データがXMLで表現される次世代の事務処理アプリケーションのモデリング手法として、本論文の提案は、多くの示唆を与える方法論である。

第3章では、1980年代中頃より始まったアプリケーション・アーキテクチャ、ソフトウェア・アーキテクチャそしてフレームワークの発展の経緯を概観した。ソフトウェア・アーキテクチャは、技術の進展と市場での栄枯盛衰を反映して、ネットワーク、OS、ミドルウェアと体系が整備されてきた。現在は、アプリケーションの分野に整備の焦点が当てられている。ビジネス・オブジェクトとかコンポーネントの部品化とか、様々な業界団体で彼らのマーケティング戦略を鎧の下に隠して、提携が為され、業界での主導権を獲得すべく、様々な活動がなされている。21世紀の早い段階でインターネット時代のビジネス・フレームワークの方向性が決まると予想している。

第4章では、1990年代中頃のマーキテクチャ（マーケティング・アーキテクチャ）の例として、筆者らの開発したオープン・ソリューション・フレームワークを説明した。このフレームワークは、現在その使命を全うしているが、この時に作成したアプリケーション処理モデル、プロダクトセットの考え方は、次のマーキテクチャである OnNet Solution[19,36]及びその次の Unisys e-@ction Solutions[37]へと引き継いでいる。

第5章では、アーキテクチャ・スタイルのような、そのスタイル間に絶対的な優劣が存在しないスタイルの選択手法を提案し、実アプリケーション・システムでの適用評価を示した。この選択手法は、アプリケーション・システム設計の初期段階に於けるアーキテクチャ設計を支援するアーキテクチャ・スタイルの選択手法としての可能性を示した。アーキテクチャ・スタイルの特性化に使用した特性軸は、情報技術の進化やビジネスの変化に対応して見直す必要がある。しかし、導入した計測尺度や選択手法は、コンピュータ業界でのシステム設計の一手法として今後の展開が期待できる。

6.2 今後の展望

21 世紀に入り、ソフトウェア・アーキテクチャの焦点は、ビジネスと密接に関連したソリューションへと遷移している。既に、ソリューション・ビジネスを普及推進するためにソリューション・アーキテクチャの標準化の活動がなされている[9]。新しいソフトウェア・アーキテクチャは、インターネットをベースにした e ビジネスと密接に関係する。筆者らは、ソリューション・アーキテクチャの標準化活動を支援しながら、Unisys e-@ction Solutions の拡充とアーキテクチャ・スタイルを利用してプロダクトセットの展開を図っている。

アーキテクチャ・スタイルとプロダクトセットの新たな展開として、CMU の SEI グループのプロダクトライン(Product Line)研究にアーキテクチャ・スタイルとプロダクトセットを提案した[45]。プロダクトラインの議論に、新たな見識を与えることができ、今後の進展が期待できる。

また、オープン・ソリューション・フレームワークで開発したアプリケーション処理モデルの一つの適用方向として、金融業界で話題になっている STP(Straight Through Processing)プロセスへの展開を、筆者らは検討している[100,101]。基幹システム系への新しい提案になり、一部の先進企業では、既にその適用が始まっている。

ソフトウェア工学における要求仕様獲得の分野に、ソフトウェア・アーキテクチャのモデリング手法を適用すべく、筆者達は、要求追跡のフレームワーク作り[92,93]と要求獲得における発見と伝達のモデリング[94]を試行している。第 5 章で導入したアーキテクチャ・スタイルの特性化の手法や導入した計測尺度や選択手法の適用を検討している。

最後に、ソフトウェア・アーキテクチャを構成する個々のプロセスに目を向けたとき、プロセスの入出力データは、全て XML 準拠の方向を向いている。個々のプロセスを厳密にモデリングする手法として、本論文で提案した論理型言語が適用できると考えている。21 世紀の新しいアプリケーション・モデリング手法の一つとして発展が期待できる。

謝辞

本研究の切っ掛けになった旧電子技術総合研究所の時代から始まり、本学への進学も含め温かく見守って下さり、研究の細部に亙り、直接の懇切なる御指導を賜りました、奈良先端科学技術大学院大学 鳥居 宏次 学長に心より深く感謝いたします。

本研究を進めるに当たり、暖かいご支援と適切な御指導を賜りました、 奈良先端科学技術大学院大学情報科学研究科 松本 健一 教授に心より深く感謝いたします。

本研究を進めるに当たり、貴重な御指導を賜りました、奈良先端科学技術大学院大学情報科学研究科 植村 俊亮 教授に心より深く感謝いたします。

本研究を進めるに当たり、貴重な御指導を賜りました、奈良先端科学技術大学院大学情報科学研究科 関 浩之 教授に心より深く感謝いたします。

本研究を進めるに当たり、暖かいご支援と適切な御指導を賜りました、大阪大学大学院基礎工学研究科 井上 克郎 教授に心より深く感謝いたします。

本研究を進めるに当たり、暖かいご支援を頂きました、奈良先端科学技術大学院大学情報科学研究科ソフトウェア構成学講座 島 和之 助手と門田 暁人 助手、そして研究室の諸氏に心より深く感謝いたします。

本研究の早い段階での共同研究者であり先輩であり、また熱心な御指導を賜りました、元日本ユニシス株式会社の山崎 利治氏に心より深く感謝いたします。

本研究の早い段階での共同開発者である元日本ユニシス株式会社の外山 晴夫氏と岩田 裕道氏および日本ユニシス株式会社の岡田 寿氏に心より深く感謝いたします。

本研究を進めるに当たり、拙い私の英文を洗練してくれた米国ユニシス株式会社の友人である Tom Turba 氏に深く感謝いたします。また、挫けそうになる私の相談相手でありました研究仲間でもある日本ユニシス株式会社の四居 雅章氏、柳田 和幸氏、妻木 俊彦氏に心より深く感謝いたします。

本研究を進めるに当たり、本業の遂行に多くの配慮と知恵を頂いた日本ユニシス株式会社の水丸 晴雄氏に心より深く感謝いたします。また陰になり日向になり励まし支援してくれた日本ユニシス株式会社の多くの仲間、そして大学時代からの友人である株式会社ケーテック・システムズ 加藤 哲彦氏に心より深く感謝いたします。

最後に、会社生活と学生生活の二足の草鞋を履き、時間的にも経済的にも、そして精神的にも厳しかった４年間を、暖かく見守り励ましてくれた妻 光迪子および綾子と千尋そして史世の３人の娘に心より深く感謝いたします。

参考文献

[1] 雨宮真人, 丸山充:関数型言語 Valid による在庫管理システムの記述, *情報処理*, Vol.26, No.5, pp.506-520, 1985.5.

[2] 青山幹雄ほか：特集：e-ビジネスを実現するソフトウェアサービス技術, *情報処理*, Vol.42 No.9, pp.855-895, 2001.9.

[3] 朝日新聞社：知恵蔵, *朝日新聞社*, P.1514, 1999.1.

[4] Len Bass, Paul Clements and Rick Kazman: Software Architecture in Practice, *Addison-Wesley*, P.452, 1998.

[5] Len Bass and Rick Kazman: Architectural-Based Development, *SEI*, CMU/SEI-99-TR-007, 1999.

[6] Alex Berson: Client/Server Architecture, *McGraw-Hill*, P.452, 1992.

[7] Kathy A. Bohrer: Architecture of the San Francisco frameworks, *IBM System Journal*, Vol.37 No.2, pp.156-169, 1998.

[8] James Cassell.: The Total Cost of Client/Server: A Comprehensive Model, *A Gartner Group Conference on the Future of Information Technology Industry*, P.19, 1994.11.

[9] 電子情報技術産業協会：ソリューションビジネスに関する調査研究報告書, *電子情報技術産業協会*, 01-シ-1, P.266, 2001.3.

[10] Mary Jo Foley: 10 architectures that will boost your business, *Systems Integration*, pp.26-42, 1990.6.

[11] 富士通（株）システム本部インターネットソリューション推進室ビジネス推進部：特集 I Internet ビジネスソリューション インターネットがビジネスを変える, *富士通ジャーナル*, Vol.26 No.2, pp.6-14, 2000.5.

[12] 二村良彦：プログラム設計法 PAD/PAM, *情報処理*, Vol.25, No.11, pp.1237-1246, 1984.11.

[13] HP 社の E-service のホームページ：URL: http://www.hp.com/solutions1/e-services/

[14] George Hutfilz：IBM が目指すオープンシステム, *ACCESS*, pp.4-19, 1992.秋.

[15] IBM: Open Blueprint Introduction, *IBM Corporation*, P.37, 1994.4.

[16] IBM: A Guide to OPEN CLIENT/SERVER, *Open Enterprise group of IBM Europe in Basingstoke UK*, 1994.

[17] IBM: Open Blueprint Technical Overview, *IBM Corporation*, GC23-3808-2, P.88, 1996.12.

[18] IBM 社の e-business のホームページ：URL: http://www.ibm.com/e-business/

[19] 今江泰：オンネット時代のソリューション・コンセプト - 「OnNet Solution」，*えすぷり - Strategy21*，Vol.79，pp.38-49，1999.秋号.
URL: http://www.unisys.co.jp/PDF/ESPRIT/79/esp79-5.pdf

[20] ISO/IEC JTC1 N1335: TSG-1 on Standards necessary to define Interface for Application Portability (IAP) Final Report, 1991.5.

[21] ISO/IEC 9126:1991: Information technology - Software product evaluation - Quality characteristics and guidelines for their use, P.13, 1991.12.

[22] ISO/IEC 10026-1: Information technology - Open Systems Interconnection - Distributed Transaction Processing - Part 1: OSI Transaction Processing Model, 1992.12.

[23] ISO/IEC T R 14252: 1996-13-15: Information Technology - Programming Languages, their environments and system software interfaces - Guide to the POSIX Open System Environment (OSE), P.261, 1996.12.

[24] 伊藤利郎, 坂和磨編著：リエンジニアリングで会社を集合天才に変える本, *オーム社*, P.340, 1995.3.

[25] JIS X 0129-1994: ソフトウェア製品の評価－品質特性及びその利用要項, P.13, 1994.

[26] Ralph E. Johnson，中村宏明，中山裕子，吉田和樹：パターンとフレームワーク，*共立出版*，P.264，1999.6.

[27] 片岡雅憲，金藤栄孝，宮本和靖，山野紘一：標準構造に基づく系統的ソフトウェア設計法，*情報処理*，Vol.25, No.11, pp.1220-1227, 1984.11.

[28] 片山卓也：属性文法による在庫管理システムの記述，*情報処理*，Vol.26，No.5，pp.478-485, 1985.5.

[29] 加藤潤三，染谷誠，板倉教，森澤好臣，山崎利治：仕様記述法の味見，*情報処理学会ソフトウェア工学研究報告*，85-SE-40-12，pp.67-72, 1985.2.

[30] Rick Kazman, et.al.: Scenario-Based Analysis of Software Architecture, *IEEE Software*, Vol.13 No.6, pp.47-55, 1996.11.

[31] Rick Kazman, et.al.: The Architecture Tradeoff Analysis Method, *Proceedings of 4th International Conference on Engineering of Complex Computer Systems (ICECCS98)*, 1998.8.

[32] 岸知二: ソフトウェアアーキテクチャに関する考察～AHP を活用したアーキテクチャ選択～, *情報処理学会ソフトウェア工学研究報告*, 2001-SE-130-14, pp.101-108, 2001.

[33] 小池晋一, 千葉修 : ソリューションフレームワーク, *NEC 技報*, Vol.52 No.1, pp.3-8, 1999.1.

[34] 久保未沙 : 複合設計, *情報処理*, Vol.25, No.9, pp.935-945, 1984.9.

[35] 久世和資 : ストリームを扱う言語 Stella による在庫管理システムの記述, *情報処理*, Vol.26, No.5, pp.497-505, 1985.5.

[36] 水丸春雄 : オンネット時代の企業情報システム構築に向けて 日本ユニシスからの提言, *えすぷり - Strategy21*, Vol.79, pp.26-37, 1999.秋号.
URL: http://www.unisys.co.jp/PDF/ESPRIT/79/esp79-4.pdf

[37] 水丸晴雄 : ビジネス即応型企業情報システムの実現を目指して, *Unisys 技報*, Vol.20 No.2, pp.167-185, 2000.8.

[38] 森澤好臣, 鳥居宏次 : 仕様から Prolog プログラムを作成する一手法, *情報処理学会第 27 回全国大会*, 3C-8, pp.507-508, 1983.10.

[39] 森澤好臣 : 構文と意味の記述によるプログラムの作成, *情報処理*, Vol.25, No.11, pp.1255-1260, 1984.11.

[40] 森澤好臣, 岩田裕道, 外山晴夫 : 分散処理システムの処理モデルの一提案, *情報処理学会ソフトウェア工学研究報告*, 96-SE-109, pp.17-24, 1996.5.

[41] 森澤好臣, 岩田裕道, 外山晴夫 : クライアント／サーバ・システム構築のためのオープン・ソリューション・フレームワーク, *Unisys 技報*, Vol.16 No.2, pp.15-33, 1996.8.

[42] 森澤好臣 : アーキテクチャとマーキテクチャの挟間で, *情報処理学会ソフトウェア工学研究会 ウインターワークショップ・イン・恵那論文集*, pp.33-34, 1998.10.

[43] Yoshitomi Morisawa, Hisashi Okada, Hiromichi Iwata and Haruo Toyama: A Computing Model for Distributed Processing Systems and Its Application, Proceeding of 1998 Asia Pacific Software Engineering Conference, pp.314-321, 1998.12.

[44] 森澤好臣, 鳥居宏次 : 分散処理システムのアーキテクチャ・スタイル, *情報処理学会ソフトウェア工学研究報告*, 99-SE-122, pp.9-16, 1999.3.

[45] Yoshitomi Morisawa: A Computing Model of Product Lines for Distributed Processing Systems, its Product Sets, and its Applications, *Proceedings of the First Software Product Lines Conference (SPLC1)*, pp.371-394, 2000.8.
This proceedings is published as; Patrick Donohoe: SOFTWARE PRODUCT

LINES - Experience and Research Directions, *Kluwer Academic Publishers*, 2000.8.

[46] Yoshitomi Morisawa and Koji Torii: A practical method to select an architectural style of product lines for distributed systems, *Proceedings of the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*, pp.1391-1397, Las Vegas, Nevada, USA, 2001.6.

[47] Yoshitomi Morisawa and Koji Torii: An Architectural Style of Product Lines for Distributed Processing Systems, and Practical Selection Method, *Proceedings of Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9)*, pp. 11-20, Vienna Austria, 2001.9.

This paper is also published as: Yoshitomi Morisawa and Koji Torii: An Architectural Style of Product Lines for Distributed Processing Systems, and Practical Selection Method, *ACM Software Engineering Notes*, Vol.26, No.5, pp.11-20, 2001.9.

[48] Yoshitomi Morisawa, Katsuro Inoue and Koji Torii: Architectural Styles for Distributed Processing Systems and Practical Selection Method, Submitted to *Information and Software Technology*.

[49] 中谷多哉子，青山幹雄，佐藤啓太，編：ソフトウェアパターン，*共立出版*，P.281，1999.11

[50] 根来龍之，國領二郎，永峯弘毅，繁野高仁，森澤好臣，吉岡正：パネルディスカッション－情報技術の戦略的意義，ユニシス・ニュース，第 410 号，1995.6.
（「手島歩三, 小池俊弘, 大熊新仁: やわらか情報戦略ブック, *オーム社*, P.272, 1996.3」に全文引用あり）

[51] 日本 IBM：Open Blueprint 入門　分散アプリケーションのためのガイド (第二版)，*日本 IBM*，P.51, 1995.9.

[52] 日本 IBM 社のパーベイシブ・コンピューティングのホームページ：URL: http://www.jp.ibm.com/e-business/pvc.html

[53] 日本ユニシス：Unisys Architecture 概要，*日本ユニシス*，P.34，1993.2.

[54] 日本ユニシス：UA 解説書 第 1 巻アプリケーション・サービス(A&IS)，*日本ユニシス*，P.342, 1995.2.

[55] 日本ユニシス：UA 解説書 第 2 巻インフォメーション管理サービス(IMS)，*日本ユニシス*, P.260, 1995.2.

[56] 日本ユニシス:UA 解説書 第3巻分散システム・サービス(DSS), *日本ユニシス*, P.237, 1995.2.

[57] 日本ユニシス:UA 解説書 第4巻システム間接続サービス(SCS), *日本ユニシス*, P.312, 1995.2.

[58] 日本ユニシス:UA 解説書 第5巻システム管理サービス(SMS), *日本ユニシス*, P.240, 1995.2.

[59] 日本電子工業振興協会：コンピュータの異機種環境利用に関する調査報告書, *日本電子工業振興協会*, 63-C-590, P.205, 1988.3.

[60] 日本電子工業振興協会：コンピュータの異機種環境利用に関する調査報告書, *日本電子工業振興協会*, 89-計-7, P.149, 1989.3.

[61] 日本電子工業振興協会：コンピュータの異機種環境利用に関する調査報告書, *日本電子工業振興協会*, 90-計-7, P.215, 1990.5.

[62] 日経コンピュータ：野村証券が基幹システムを刷新へパソコン1人1台で業務改革に挑戦, *日経コンピュータ*, 1995.5.13 号, pp.54-55, 1995.5.

[63] 西村高志 : MOTHER SYSTEM によるソフトウェアの設計と製造, *情報処理*, Vol.25, No.11, pp.1247-1254, 1984.11.

[64] Barry D. Nusbaum, Thomas Liu, Marco Pistoia and Giancarlo Rochester: IBM Network Computing Framework for e-business Guide, *IBM Corporation*, P.314, 1998.9.

[65] 大木優, 二村良彦, 竹内彰一, 宮崎敏彦, 古川康一：論理型並列プログラミング言語 Concurrent Prolog による在庫管理システムの記述, *情報処理*, Vol.26, No.5, pp.469-477, 1985.5.

[66] 大野侑郎：ジャクソンシステム開発法, *情報処理*, Vol.25, No.9, pp.955-962, 1984.9.

[67] 岡田康治 : 階層的仕様言語 HISP に基づく変換法プログラミング, *情報処理*, Vol.25, No.11, pp.1261-1268, 1984.11.

[68] 岡本茂, 松山泰男, 大島邦夫, 編 : 精説コンピュータ理工学事典, *共立出版*, P.861, 1997.7.

[69] OMG: Business Object Domain Task Force RFI-1, *Object Management Group*, OMG Document bom/97-06-02, P.17, 1997.6.

[70] Open Group: The Open Group Architectural Framework (TOGAF) Version 6, *the Open Group*, P.263, 2000.12.
URL: http://www.rdg.opengroup.org/public/arch/index.htm

[71] オラクル社のホームページ：URL: http://www.oracle.com/

[72] Fernando C. N. Pereira and David H.. Warren: Definite clause grammars for language analysis, *Artificial Intelligence*, Vol.13, pp.231-278, 1980.

[73] Luis M. Pereira, Fernando C. N.. Pereira and David H. Warren: User's Guide to DECsystem-10 Prolog. *Div. De Informatica*, LNEC, Lisbon and Department of AI, University of Edinburgh, P.60, 1978.9.

[74] 齋藤真人，花塚光博，尾山壮一，稲葉慶一郎：これからの情報システム"Network Objectplaza"，*日立評論*，Vol.80 No.5，pp.9-14，1998.5.

[75] SAP 社のホームページ：URL: http://www.sap.co.jp/

[76] SEER Technologies: Client/Server Infrastructure Road Map, *Client/Server Journal*, ComputerWorld, 1994.

[77] Mary Shaw and David Garlan: Software Architecture, *Prentice Hall*, P.242, 1996.

[78] Mary Shaw and Paul Clements: A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems, *Proceedings of 21st International Computer Software and Application Conference (COMPSAC'97)*, pp6-13, 1997.8.

[79] John J. Shedletsky and John J. Rofrano: Application reference designed for distributed system, *IBM Systems Journal*, Vol.32, No.4, pp.625-646, 1993.

[80] 芝山悦哉，松田裕幸，米澤明憲：並列オブジェクト指向言語 ABCL による在庫管理システムの記述，*情報処理*，Vol.26, No.5, pp.460-468, 1985.5

[81] 調重俊：インターネット時代のシステムインテグレーション，*東芝レビュー*，Vol.54 No.1, pp.30-32, 1999.1.

[82] サン社のホームページ：URL: http://www.sun.com/

[83] 鈴木君子：構造化プログラミング ワーニエ・メソド，*情報処理*，Vol.25，No.9, pp.946-954, 1984.9.

[84] 高野みさこ，檀原可一，米井達哉，萩野美穂：日立製作所の新ソリューション体系"Cyberfront"－時代を勝ち抜く情報戦略企業を支えるために－，*日立評論*，Vol.80 No.9, pp.4-8, 1998.9.

[85] ドン・タプスコット，アートキャストン著（野村総合研究所訳）：情報技術革命とリエンジニアリング，*野村総合研究所*，P.514，1995.4.

[86] Tetsuo Tamai: How Modeling Mehtods Affect the Process of Architectural Design Decisions: A Comparative Study, *Proceedings of 8th International Workshop on Software Specification and Design*, pp.125-134, 1996.3.

[87] 手島歩三，小池俊弘，遠藤清三：ソフトウェアのダウンサイジング，*日本能率協会マネジメントセミナー*，P.261，1994.10.

[88] 所真理雄，松岡聡，垂水浩幸，編：オブジェクト指向コンピューティング，*岩波書店*，P.442，1993.11.

[89] Koji Torii, Yoshitomi Morisawa, Yuji Sugiyama and Tadao Kasami: Functional Programming and Logical Programming for the Telegram Analysis Problem, *Proceedings of the 7th International Conference on Software Engineering*, pp.463-472, 1984.3.

[90] 鳥居宏次，嵩忠雄，杉山裕二，森澤好臣：Logical Programming for the Telegram Analysis Problem，*情報処理学会ソフトウェア工学研究報告*，86-SE-46，pp.41-48，1986.2.

[91] Koji Torii, Yuji Sugiyama, Mamoru Fujii, Tadao Kasami and Yoshitomi Morisawa: Logical Programming for the Telegram Analysis Problem, *Computer Languages*, Vol.12, No.1, pp.9-20, 1987.1.

[92] 妻木俊彦，森澤好臣：要求追跡の枠組みと支援ツール，*情報処理学会 OO2000 シンポジウム*，pp.125-132，2000.8.

[93] Toshihiko Tsumaki and Yoshitomi Morisawa: A Framework of Requirements Tracing using UML, *Proceedings of 2000 Asia Pacific Software Engineering Conference*, pp.206-213, 2000.12.

[94] 妻木俊彦, 森澤好臣:要求獲得における知識の発現・伝達モデル, *情報処理学会 OO2001 シンポジウム*，pp.33-40，2001.8.

[95] Unisys: Integrated Information Environment: A Conceptual Overview of the Unisys Architecture (7833 1667-100), *Unisys Corporation*, 1994.3.

[96] Unisys: Open Technology Framework. Unisys Corporation, P.51, 1996.7.

[97] 臼井義美:「SP-FLOW」によるデータ構造に基づくシステム設計法, *情報処理*, Vol.25, No.11, pp.1228-1236, 1984.11.

[98] 矢島竜司，桝渕吉弘，塩野入匡宏：iBestSolution の概要，*NEC 技報*，Vol.53 No.8, pp.5-8, 2000.8.

[99] 山崎利治：共通問題によるプログラム設計技法解説，*情報処理*，Vol.25，No.9, pp.934-934, 1984.9.

[100] 柳田和幸，関沢賢，中川彰男，高宮康次：イントラネットによる知識活用の仕組み，*情報処理学会ソフトウェア工学研究報告*，98-SE-120，pp.1-8，1998.7.

[101] 柳田和幸，四居雅章，森澤好臣：メッセージ駆動型システムの提案，*情報処理学会ソフトウェア工学研究報告*，99-SE-124，pp.73-80, 1999.10.

[102] 吉成安宏：SOLUTIONVISION が拓く次世代企業情報システム，*富士通ジャーナル*，Vol.24 No.2, pp.6-11, 1998.7.

[103] 湯浅太一：モジュラ・プログラミング言語イオタによる在庫管理プログラムの仕様記述とプログラム記述，*情報処理*，Vol.26, No.5, pp.486-496, 1985.5.