政府情報システムのシステム刷新に関する調査研究

平成21年3月

社団法人　行政情報システム研究所

まえがき

コンピューター・システムの世界では 90 年代中ごろから徐々にメインフレームからオープン化の動きが活発化してきた。当時、一般的に高コストであったメインフレームから、PC サーバーへの移行はシステム・コストを下げる一定の効果があり、インターネットの拡大とともに IT 化への流れを強めた一因であった。

このメインフレームから PC サーバーへの移行は、システム・コストを押し下げる一方、メンテナンスや導入等に掛かる人手は増加し、それに伴い発生するコストは見えにくいコストとして運用経費を増加させ、IT 要員への過大な負荷も問題となっている。また、PC サーバーとそのソフトフェアの不安定性からくる、システム全体の信頼性にも様々な対応が徐々にされているものの、依然としてオープン化の課題として残っているのが現状である。

そのような IT 環境をめぐる変化の中、政府においても「電子政府構築計画」の下、各府省において「レガシーシステム見直しのための行動計画（アクション・プログラム）」が策定され、情報システムの見直しが進められてきている。しかしながら、システムに関する視点のない評価や、最新の技術動向や将来動向を反映しないシステム構成は結果として大きなコスト増を招きかねず、単純なコスト削減だけを考えた見直しには課題が多い。

このような背景の下、政府の情報システムを刷新するに当たり、技術面を含む幅広い視点から最適システムを選定するための検討方法、及びコスト管理等のガバナンスに関わる課題について、問題の構造や原因の調査研究を行った。そのことにより、実際に受け皿となるべき技術や移行のシナリオの実効性を検証し、是正すべきポイントについても検討を行った。そして、「政府情報システムの刷新化」が本来の意義に沿って、時流に適った妥当な施策とするための指針となり、効率的な電子政府の推進に資することを目的として、本調査研究を実施したものである。

なお、本調査研究は、ガートナージャパン株式会社の協力を得て、当研究所において実施したものである。

また、本調査研究の実施にあたっては、過去の文献調査や当研究所における知見だけではなく、当研究所会員企業の中でメインフレームの製造販売を実施している企業の参画を得て、当研究所内に「行政における基幹系システム・フレームワークに関する研究会」を設置し、最新技術動向や政府情報システムの直面している最新状況の把握をヒアリングなどを実施しつつ調査研究を行った。研究会参加企業及び参加者は次ページ掲載の通りであり、ご協力に感謝申し上げる次第である。

社団法人　行政情報システム研究所

＜行政における基幹系システム・フレームワークに関する研究会　参加者名簿＞

大澤　　暁	日本アイ・ビー・エム株式会社	公共事業．クライアントIT推進エグゼクティブITアーキテクト
広上　覚	日本電気株式会社	官公ソリューション事業本部副事業本部長
甲斐　隆嗣	株式会社日立製作所	全国公共システム本部公共システム推進第二部長
遠藤　明	富士通株式会社	官公庁ソリューション事業本部　本部長代理 （兼）公共SBG　ソリューション開発センター　センター長

（肩書きは平成 21 年 6 月現在　敬称略）

目次

1. 本研究の背景

1.1 情報システムの刷新化とは

1.1.1 情報システムの刷新

政府における情報システムの歴史は古く、1950 年代までに遡る。当初は主に、気象業務や統計集計等の技術計算用途が目的[1]であったが、徐々に厚生年金、保険計算や給与等の計算業務における省力化を目的として導入された。1960 年代には、第一次情報化ブームと呼べる情報化投資の波が押し寄せ、コンピューターを活用した個々の業務を効率的に集計するために導入されたオンラインシステムへと発展する。この後、OA と呼ばれた文書処理や表計算ソフトウェアの導入、そしてインターネットの普及によりインターネットを活用した業務アプリケーションを活用するに至っている。

この技術発展に対応して、官民問わず情報システムの刷新は幾度となく行われてきた。情報システムの刷新とは、機能もしくはハードウェアが老朽化し、十分な機能や性能を発揮できなくなったシステムや、老朽化が原因でシステムの運用コストが増大し、そのコスト構造に手を付けられることなく高止まりしたまま運用し続けているシステムについて、それらの改善を目的とした廃止及び廃棄を伴う新規導入を行うことを指している。

政府における情報システムの刷新は、メインフレームの刷新が主眼におかれているケースが多い。メインフレームは、企業及び社会システムにおける大規模なコンピューターであり、独自のアーキテクチャをベースに設計されたコンピューター本体及びその周辺機器の呼称である。日本国内では汎用コンピューターとも言われている。堅牢でかつ安定的に動作することを特徴としており、政府においても 1970 年代から長期にわたって使用し続けているアーキテクチャであるため、1990 年代に台頭してきたオープン系アーキテクチャのハードウェアと対比し、後述するように「レガシーシステム」と呼ばれ、情報システム刷新の象徴とされてきた。

---

[1] 参考：奥村裕一(2007)／行政事務機械化研究協会(1965)。

1.1.2　政府による「レガシーシステム見直しのための行動計画」

中央政府や地方自治体においても、1990 年中頃より UNIX[2]等をプラットフォームとして採用したオープン系コンピューター・システム[3]がハードウェア価格性能比の向上や、クライアントサーバモデル、TCP/IP 等の標準プロトコルなどの技術発展等を背景に普及してきた。既存のメインフレームは（オープンシステムとの対比で）次第にレガシーシステムとも呼ばれるようになった。加えて、システムの老朽化や保守コスト及び要員ノウハウ継承の問題、またハードウェアとソフトウェアを分離調達できないこと等によるベンダーロックインの問題等、様々な課題が指摘され始めていた。

このため、2003 年 3 月 25 日に、自民党 e-Japan 重点計画特命委員会が「レガシーシステム改革指針」を発表し、①業務の継続性を重んじるあまり、巨大なシステムが長年にわたり非競争環境で調達されてきたこと、②また、その維持運営の費用の増大で、戦略的な IT 投資に予算が振り分けられにくい構造にあることを指摘し、早急な現在の情報システム及びその調達の見直しを行うことを迫った。

この見直しを行うことで、「長年にわたって随意契約によりシステムの改良、保守を繰り返しているような旧式のシステムについては、価格性能比の向上した新たなシステムを構築することによって、結果的にシステムの費用対効果は格段に改善される」ということが提唱された。

このような状況を踏まえ、政府は電子政府構築計画[4]の推進の一環として、業務及びシステム最適化計画を策定する方針を打ち出し、各省庁にてシステム最適化計画を策定すること、特にレガシーシステムに該当するものについては、「レガシー

---

[2] 1968 年に AT&T 社のベル研究所で開発された OS を起源とし、企業や教育機関、官公庁で高い安定性を必要とされるアプリケーションで用いられる。C 言語というハードウェアに依存しない移植性の高い言語で記述されたこともあり、多くのプラットフォームに移植され、また、多くの派生形が誕生している。Linux はその一つである。

[3] システム構築環境において、オープン標準（開示された技術）に準拠したソフトウェア及び技術や、使用しているシステムを指す。Windows は開示された技術標準ではないため、定義上はオープンシステムではないが、多くの場合オープンシステムと呼ばれている。

[4] 2003 年（平成 15 年）7 月 17 日に各府省情報化統括責任者連絡会議（CIO 連絡会議）において決定された電子政府構築に係る政府の具体的な取り組み計画をまとめたもの。2010 年までの政府による具体的な電子政府への取り組みについて記述されている。2004 年 6 月 14 日にこれまでの進捗状況や課題を反映させた改定が行われている。

システム見直しのための行動計画」を 2003 年 7 月に提示し、これに基づき必要な見直し（オープン化を前提とした、主に技術的な観点での刷新の可能性調査）を行うこととなった。

この時に見直しの対象となったのは、①中央省庁において、年間 10 億円以上の経費を要する情報システム、②汎用コンピューター、オフコン（開発業者独自のオペレーティングシステムを搭載した中型コンピューター）を使用したシステム及びこれらに接続するためのシステム、③1994 年（平成6年）以降随意契約が継続しているシステム、とされた。その結果、対象となったのが以下のシステムである。

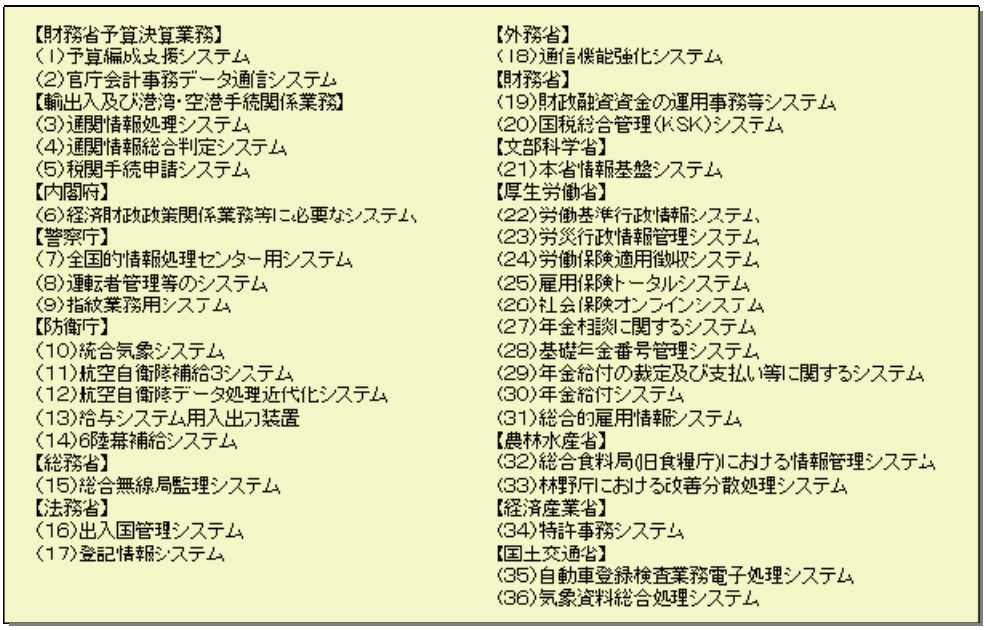


図 1　政府「電子政府構築計画」における対象レガシーシステム

この定義が官公庁におけるレガシーシステムとして扱われているが、本稿で扱うレガシーシステムは、基本的には図1に示されたシステムを含む汎用コンピューター、オフコンとする。但し、図1のシステムはあくまでも見直しを検討する対象のシステムであり、必ずしも刷新を要請されたものではなかったことには留意が必要である。対象となったシステムにおいては、刷新可能性調査を実施したうえで、刷新の方向性を検討する余地が与えられていた。また、「メインフレーム＝レガシーシステム」と語られることが多いが、本稿で用いるメインフレームという用語は、前述したように独自のアーキテクチャーをもったコンピューター・システムの呼称であり、レガシーシステムを構成している一要素である。したがってレガシーシステムと同義ではないこと

にも留意いただきたい。これは官民問わずIT業界における一般的な用語の定義である。

1.1.3 民間で起こっているシステム刷新の流れ

一方、民間企業では、メインフレームが負の遺産として問題視されることもあったが、それよりも、オープンシステム化の流れとして、企業が対応しなければならないシステムの進化やシステムの刷新の選択肢のひとつとして捉えられていた傾向が強かった。

前述した通り、1990年中頃よりUNIXが可用性と信頼性を増してきたことで、企業活動のプラットフォームとして利用できる「オープン系コンピューター」として認知され始め、徐々に企業の基幹システムの中核を担うようになった。この流れの中でオープン化は北米を中心にダウンサイジングとしてもてはやされ、システム構築においての一つの潮流となった。この時に大手メインフレームベンダーは大幅な事業構造の転換を迫られた。

日本においても、1995～2000年ごろにかけて「連結経営」「グローバルスタンダード」等というキーワードと共に、主に財務的な視点において経営資源を統合的に管理し、有効な配分等を計画しながら経営の効率化を図るための手法・概念を実装するというコンセプトが盛んに宣伝された。その結果、このコンセプトを具現化したソフトウェアパッケージであるERPの導入が一種のシステムトレンドとして現れ始めた。特に大企業が、会計系等の基幹システムで使用されていた中小規模のメインフレームを、一斉に欧米系のパッケージに切り替わる流れとなっていったのである。当時、このパッケージにはプラットフォームとしてUNIXを採用することが多かったが、国内においてUNIXを用いた大規模基幹オープンシステム導入の実績は限られており、特にパフォーマンスの面でいくつかの失敗が見られた。これらの失敗は、ハードウェアの冗長化技術やデータベースの負荷分散技術といった可用性向上の取り組みを加速し、現在のオープン系技術における高信頼性に結びついている。

今日では、特に安定性や信頼性の面においては、オープン系技術を駆使してのメインフレームの刷新は困難でなくなっており、移行手法及び移行目的に合わせて多数のハードウェアやソフトウェアの選択肢が用意されることとなった。

一方で、オープン環境への移行を検討、もしくは移行を実施したケースの中には、現行のプログラム資産の棚卸をした結果、オープンシステムへの移行そのものの中止や、アプリケーションの部分的な移行に留まるケースも存在する。

例えば、絶対に停止できない金融機関の勘定系システムや公共性の高い社会インフラの管理システム等においては、未だにメインフレームが継続して選択されることも多い。また、当然のことではあるが、オープン技術の発展と同じく、メインフレーム関連技術も進化しており、Linux プラットフォームを選択できるメインフレーム（補足1参照）にアップグレードすることで、既存資産の活用というメリットを享受しながら、オープン技術の利用を可能とする方式を採用するユーザーも存在する。

このように、民間における情報システム刷新では、オープン化ありきではなく、目的や状況に合わせて、いわば適材適所で移行している。システムの更改に際しては、部分的にメインフレームをアップグレードして継続利用するケースや、完全にメインフレームを継続しながら、周囲のオープンシステムと連携することでシステム機能を向上するケース等、幅広い意味での刷新が行われているのである。

1.1.4 「レガシーシステム見直しの行動計画」のその後

上記民間事例と比較すれば、政府における昨今のレガシーシステム見直しや最適化計画は、調達の透明性向上やコストの適正化のみではなく、適切なシステムアーキテクチャへの移行も期待されているはずである。

現に最適化計画では、適切なアーキテクチャ構築を標榜し、EA（エンタープライズアーキテクチャ）の手法を用いている。EA では政策・業務体系を整理したうえで、データ体系、適用処理体系、そしてそれを支える技術体系の検討を実施することとなっている。しかしながら、調達時においては、仕様書等においてアーキテクチャいう概念は用いられていない。また、オープン化の必要性や手法の妥当性について、メインフレームを継続利用する場合との比較、検証を十分に行うことなく単にメインフレームからオープン系コンピューターへの移行を前提としているケースがほとんどであると思料される。

最適化計画やEAはオープン化のための方法ではなく、あくまで業務とシステムの関係を体系化し、可視化する手段であり、組織全体としてのIT採用方針を決定付けるはずである。もし、その通りに実施されていれば、メインフレームを活用したレガシーシステム刷新という選択肢も存在したはずであるが、そのような最適化はごく少数である。

このようなオープン化の流れは、部分的な分離調達を実現したことで、新規事業者の参入を促す等の効果も一部には見られる。しかしそれによって、調達や開発の局面においては、複数の事業者の作業をまとめながらシステム構築を進めることが必要とされ、その負荷が新たな課題として認識されつつある。

1.2　情報システム刷新における諸問題の現状

1.2.1　システムの安定性の問題

2004 年から 2006 年にかけて、ガートナー社が国内ユーザーを中心に調査した結果を見ると、一般的に最も懸念され、レガシーシステムに多く用いられているメインフレームからの移行に向けての課題は、図 2 に示す通りシステムの信頼性や安定性（図 2）である。これはユーザーのオープンシステムの信頼性に対する誤解なのか、それとも事実として未だに信頼性を十分に担保するに至っていないのか、現状を確認する必要があるのではないかと考えている。もちろん、メインフレームで必要とされる信頼性は高いレベルが求められるが、基幹系オープンサーバーにおける信頼性も格段に向上している。従って、メインフレームからの移行プラットフォームとして、十分選択肢となりうる状況にあるはずであり、この事実を明らかにする必要がある。

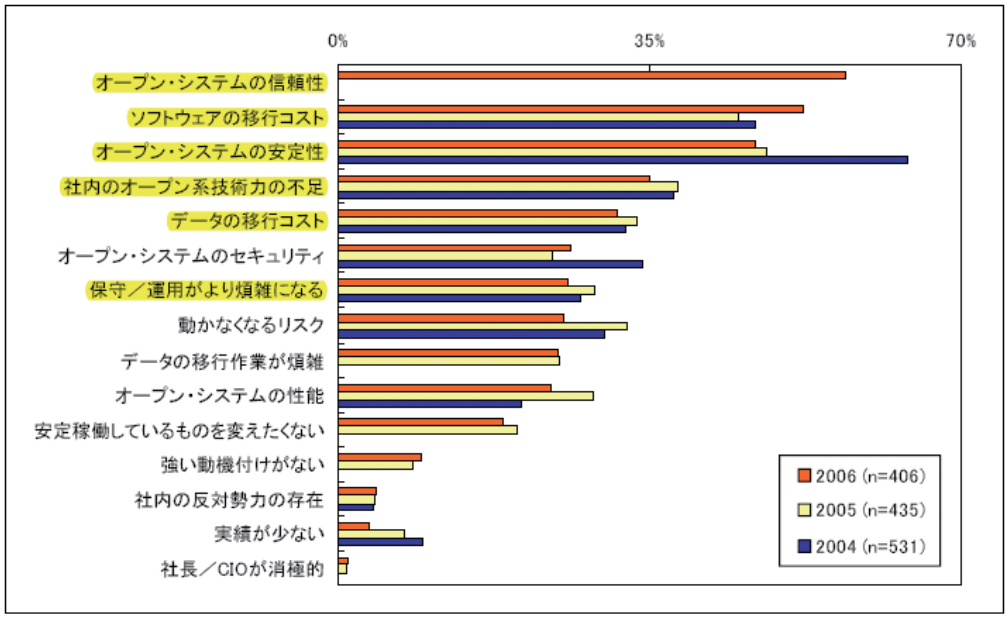


図 2　オープン化移行の懸念事項（出展：ガートナー社　2006 年 11 月）

また、単体のサーバーだけでなく、システム全体としての信頼性を向上する技術も進化しているはずであり、仮想化技術等の発展の中で生まれてきたデータベー

スクラスタ[5]技術等や、接続されたサーバーの台数を増やして処理能力を向上させるスケールアウト[6]技術等、ネットワーク上での分散処理に適した形態でかつ、信頼性を確保している技術の進展状況についても調査していくことが必要である。

更に、本調査研究においては、これらの技術が本当にメインフレームの受け皿として十分に機能しうるものなのかに関して、未だに金融業界の勘定系や公共システムにおいて広くメインフレームが採用し続けられる理由と合わせて、検証していく。

また、図 2 にあるように、「社内のオープン系技術力が不足」することで、これらオープン系技術を用いたアーキテクチャの設計を行うことに懸念が見られることや、多様なオープン技術を組み合わせて利用することで、「保守／運用がより煩雑になる」という懸念も顕在化している。オープンシステムのデメリットとして、複数のベンダの製品を組み合わせ、容易でかつ統合的にアーキテクチャを維持、管理することは、やはり煩雑なのであろうか。高い技術スキルをカバーする解は存在しないのであろうか。これらの課題に対して、現在どのような対処方法が存在するのかも検証していく必要がある。

1.2.2 コスト削減の実現性の問題

自民党 e-Japan 重点計画特命委員会の「レガシーシステム改革指針」にも言及されていた様に、この定義におけるレガシーシステムを刷新する目的は、分離調達及び競争入札の実現による運用コスト削減であり、レガシーシステム（この場合においてはメインフレームとほぼ同義）をオープン環境への移行を行うことによって運用コストが大幅に下がると同時に柔軟性を確保できるという意見がベースとなっている。

---

[5] データベースサーバ複数台で 1 つの DB システムとして構成することで、飛躍的な可用性と拡張性を実現するクラスタリング方法。障害時にダウンしたサーバーを引き継ぎ、残りのサーバーに切り替わりサービスを継続できる。また、処理を行うサーバーリソースが足りなくなったら、サーバーを追加することで処理能力を追加できる。データベースソフトウェアベンダー各社によって実現方法は異なる。

[6] CPU やメモリのような、筐体内でできるアップグレードではなく、サーバー等の筐体そのものを追加し、パフォーマンスを向上したり、信頼性を向上するという思想である。多数のサーバーを連携させて動作することで、保守や障害発生時にもサービスを完全に停止させることなく、故障機の交換や追加ができることが利点であるが、半面、サーバーライセンス等が光学になるという弱点もある。

しかしながら、先に挙げた図 2 においては、ユーザーのメインフレームオープン化の懸念としてユーザーが 2 番目に挙げているのは移行コストである。ユーザーとしては、たとえ運用コストが安くなるとしても、移行のコストが増大であるのならば、複数年で考えても投資したコストが回収できず、結果的に投資がかさむだけになる、と冷静に判断している可能性があるのかもしれない。

また、一般的にコスト削減とは、システム自身の削減は行わず、既存システムと同等の機能を非機能性能の担保とともに、現状よりも安価なコストにて運用可能となる状況を実現することを目的としている。しかしながら、調達においては初期費用としてのハードウェア価格の下落によるコストメリットを確認できていても、将来にわたる運用コスト、例えば OS のバージョンアップのコスト等を明確に予測することは難しいはずである。

予測が難しい主要な要因は、既存システムの運用コストが不明確もしくは変動的であることや、職員の稼動等を含む包括的な運用コストを算出できていないこと等であると推測される。このような状況においては、「コスト削減」という目的に合致した評価項目について予め定めておき、調達前と調達後で比較し、実際のコスト削減効果を評価することが重要ではないだろうか。この点についてもガバナンス及び調達の観点から検証する必要があると考える。

そもそも、官公庁においては「レガシーシステム刷新化」が「オープン化」の実現ありきのケースも多い。メインフレームを廃止できればよい、オープン化できればよい等といった風潮が見え隠れするため、目的が単なるオープン化技術の採用なのか、それともコスト削減なのか、不明確であることが多いのではないだろうか。

更に、オープン化と分離調達を進めたが故に生じた、複数のベンダーを管理する運用やプロジェクト管理の複雑性から、見えなかった管理コストが増大しているなどの新たな課題が発生しているという意見もある。具体的には、図 3 に示すように、単一のベンダーでは生じなかった、複数のベンダーの契約管理やベンダー間の意見や仕様の調整、そして、統合的にシステムのテストを行う際のプロジェクト管理等に係る労力やコストが余計にかかってくるという主張である。（図 3）

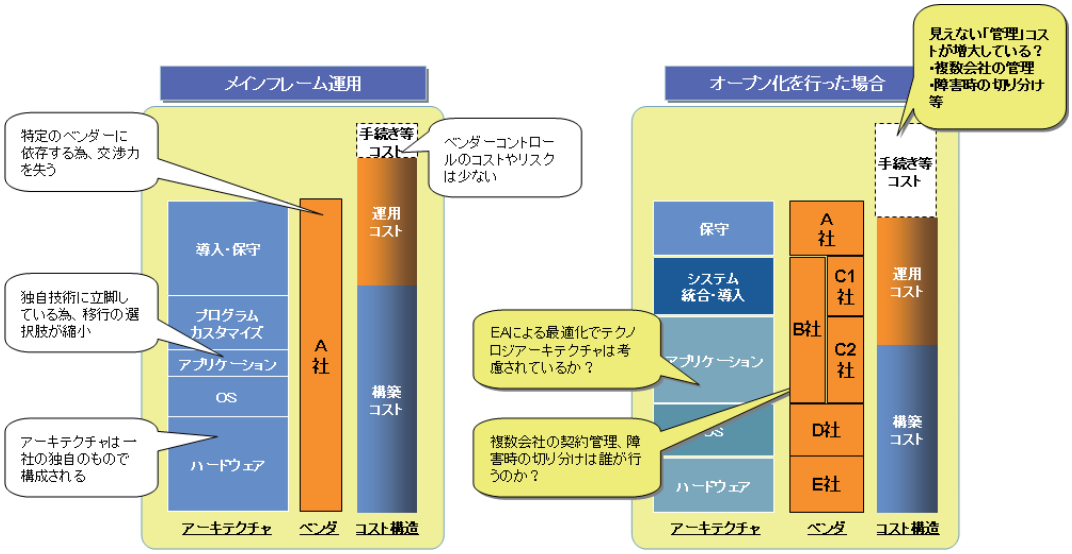


図 3　メインフレーム運用とオープン化後の IT コスト構造のモデル

刷新可能性調査の実施からすでに数年が経過しており、この間に市場においてはメインフレーム及びオープン双方の技術の格段の進歩が見られ、かつ民間等での「オープン化」事例も増加している。これまで行われてきた、「刷新しなければならない」というレガシーシステムへの数年前の取り組み方針は、果たして現在において妥当かどうか検証する必要がある。このことは、ここまでに挙げた点を検証するとともに、現在の技術の動向を把握することは重要であることを示している。

加えて、こういった問題を解決するためには技術面だけのアプローチでは不足である。ガバナンスの問題として、目的を明確にしながら、そもそも「コスト」とは何を示すかという定義や、投資に対する適切な「リターン」の基準、更には「リターン」を明確化する等の取り組み、そして導入目的に対しての効果の測定が必要である。さらに、その手法や調達のあり方についても検討していく必要がある。

1.2.3　メインフレームのオープン化が進展していない事実

ここまでに挙げた課題が理由となっているのか、民間企業においてもメインフレームの刷新が実際には思ったより進んでいないようである。この傾向を経年で示すのが図 3 である。この図を見ると、2005 年ではメインフレームからオープンへの流れが見て取れるが、2006 年に入ると継続利用が増加し、オープンへの流れが弱まっているのが分かる。

この状況から、民間企業におけるユーザー全体の傾向としては「移行を検討してみたが、やはり安定性やコストの問題を考えると移行できない」もしくは「移行を検討したが、移行する必要がなかった」という姿が浮き彫りになっているのではないだろうか。

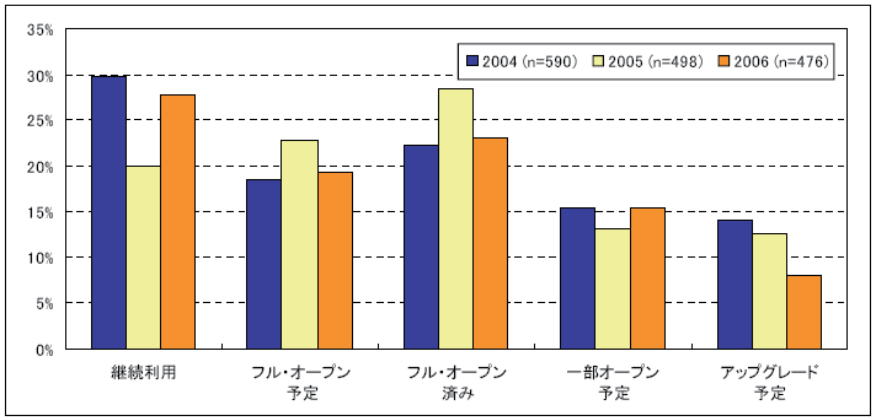


図 4　メインフレームの移行状況（出展：ガートナー社　2006 年 11 月）

以下は推測ではあるが、ここ数年の動きを踏まえると、2003 年頃にベンダーやインテグレーターから数々のメインフレーム移行手法が紹介され、数年をかけて検証を行った結果、実際にその手法で移行したケースもあると見られるが、多くは何らかの理由で採用を見送った。そのため、全体としてみると、それほどの進捗はないという結果になった。そう考えられるのかもしれない。それは図 4 に示すように、オープン化の機運がそれほど高まっていないことからも分かる。

更に図 2 及び図 4 が示したように、ユーザーは未だに安定性と移行コストに対する懸念を払拭できていないのが現状だと判断するか、もしくは、ユーザーにとって、（特に民間の）メインフレームをオープン化することは必然ではないのかもしれないと判断すべきなのかもしれない。

少なくとも、メインフレーム継続を含めて、各ベンダーはユーザーが満足する刷新のシナリオとその実行能力については十分にユーザーに訴求できていない可能性がある。その観点から言えば、本調査研究としては、各ベンダーが有するレガシーシステム移行のシナリオに関して、実行能力、特に受け皿となる技術を有しているのか、また実績があるのか調査する必要があるだろう。

実際、非常に膨大かつ緻密なプログラム群、特に階層型 DB[7]との複雑な関係を見せる COBOL[8]等において簡単に移行できる手法は存在しない。従って、単に「レガシーマイグレーション[9]」というソリューションを適用すれば済むわけではないという前提に立ち、各ベンダーが移行コストとリスクを抑え、オープン環境に移行するためのシナリオと実行能力について、改めてユーザーに訴求するべき時が来ているのかもしれない。

[7] データをツリー構造で表したデータモデルであり、データを上から下へと見ていくために親データと子データという関係でデータを体系化する。主にメインフレームで用いられていた。

[8] 事務処理用に開発されたプログラミング言語であり、誕生から半世紀ほど経っているが、未だに多くの企業や政府のコンピューターで用いられている。事務員でも使える言語として開発されたため、英語に近い記述になるようなコマンド語彙や文法（シンタックス）になっている。また過去に政府や企業の膨大なデータが COBOL によって処理され蓄積されているため、現在でも COBOL によって電算処理が処理されている情報は膨大な数にのぼると言われる。

[9] 基本的には、メインフレームで構築されたシステムを、UNIX や Windows 等のプラットフォームに移植することを指す。この名称を用いたオープンへの移行サービスは一時期市場を席巻した。

2. システム刷新を取り巻く技術及びアーキテクチャ動向

2.1 移行時における技術基盤選択肢の多様化

2.1.1 基幹系サーバー選択肢の多様化

1章の最後に述べたように、本調査研究の目的は「本来の政府情報システム刷新とはいかなるものであるか」の追求である。この目的を達成するに当たり、レガシーシステムの中核であるメインフレームの受け皿となりうるオープン系技術における基幹系サーバーの動きについて、歴史的な変化と現在の状況を検証してみたい。

2001 年、企業においてオープン化が一種のトレンドとなっていた頃に、米国インテル社は、ヒューレット・パッカード（HP）と共同開発した高性能サーバアーキテクチャである IA-64 を初めて採用した、64 ビット CPU の Itanium（アイテニウム）を発表した。この出来事は他の陣営にも、オープンハイエンドサーバーの技術競争と出荷を促すことになった。更に 2006 年には、明確にメインフレームからの置き換えを謳った Itanium の後継 CPU である Itanium2 が発表され、NEC、日立、富士通等国産主要メーカーからもオープンハイエンドサーバーがリリースされた。また、特に日本においては独自のメインフレームやオフィスコンピュータの後継機種用の CPU として採用されている。

一方、2007 年 4 月にこれまでの企業においてハイエンド領域を担っていた UNIX である SPARC 系からも、富士通と米国サンマイクロシステムズ社が共同開発した SPARC Enterprise シリーズが発表された。この SPARC Enterprise は富士通が主体となって開発した SPARC64 VI プロセッサを 4 ～64 ウェイ（8 ～128 コア[10]）搭載可能なサーバーである。仮想化機能としても、1 台のサーバーを 1CPU 単位で最大 24 の領域に区切るハードウェア・パーティショニングをサポートし、メインフレームと同等の信頼性を備えながら仮想化/統合化プラットフォームとして位置付けられた製品である。

[10] CPU の中核となり、演算処理等を行なう部分を指す。この場合、正しくは「プロセッサコア」という用法が正しい。

更に、2007 年 5 月、IBM は POWER6 搭載サーバーを発表した。この POWER プロセッサ[11]の最大の特徴は、昨今の省電力のニーズに対応するために、他社が行うマルチコア化[12]等のクロックアップ以外の手法を使うしかないという見方が主流であった中で、リーク電流問題を解決することにより、消費電力を POWER5 相当に抑えながら 4.5GHz という 高い動作周波数を実現し、性能を約 2 倍まで高めたことである。また、このプロセッサのコアに 10 進数浮動小数点（DFP）演算ユニットを搭載することで、金融計算、秒単位の通信料計算、会計業務等の商用計算の半数以上を占める DFP 演算の処理速度を飛躍的に向上させ、従来のソフトウェアによる 10 進数計算に比べて 40 ～560 倍の性能向上を可能にしている。

今日、ハイエンドの Itanium サーバー[13]と UNIX サーバー、そして POWER 搭載サーバーは、企業における信頼性及び安定性の要求されるサーバーの選択肢として安定的な地位を占めている。更に近年は x86 プロセッサ[14]においてはマルチコア化等の性能充実が著しく、x86 系のプロセッサでは最高の処理能力・処理速度を有し最上位に位置する製品である Xeon プロセッサ搭載サーバーのパフォーマンス性能の向上に伴い、ハイエンド及びミッドレンジ、更にローエンドといったオープンサーバー市場のカテゴリ間の垣根は取り払われつつある。特に、この Xeon プロセッサは出荷実績も多く、ローレンジからミッドレンジサーバー市場を席巻しているだけではなく、近年は豊富な採用実績からミッドレンジからハイエンド市場にも顔を出すようになってきている。また、多くのサーバーベンダーが Itanium より Xeon を主要な CPU と位置付けて製品開発を行っていることもあり、構成によっては上位カテゴリ製品の性能を凌ぐことも実現可能なことから、メインフレーム移行プラットフォーム先の

---

[11] IBM 社が開発・製造する RISC アーキテクチャのプロセッサ(CPU)のアークテクチャ名であり、またその製品名でもある。POWER アーキテクチャの特徴として、低いクロックで性能を発揮できるという特徴を持つ。

[12] 1 つのプロセッサパッケージ内に複数のプロセッサコアを封入した技術で, 複数の並行プロセスを同一プロセッサパッケージ内で使用することを指す。

[13] 米国 Intel 社製のハイエンドサーバー用プロセッサ Itanium を採用したハイエンドサーバーを指す。Intel 社の戦略製品であり、命令セットを IBM、HP、Sun 等の UNIX サーバー向け RISC プロセッサの代替として勧めている。

[14] 米国インテル社製のマイクロプロセッサの命令セットやアーキテクチャで構成されたマイクロプロセッサであり、これをベースとした互換性のあるマイクロプロセッサを総称して指すこともある。ここでは主に Xeon プロセッサを搭載したサーバーを想定している。

選択肢は多様化しつつある。それらを概観するために、ガートナー社の調査・分析結果を基に整理した結果を図 5[15]に示す。

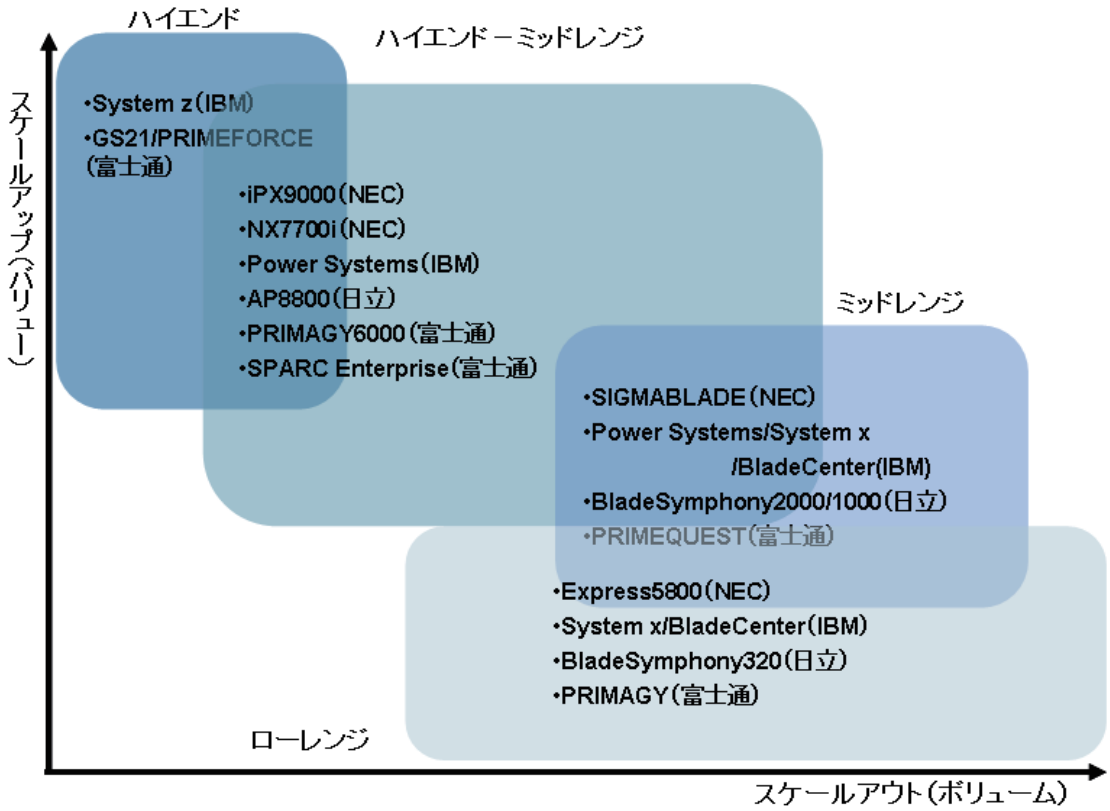


図 5　国内主要ベンダーのサーバー製品

この多様化する選択肢の中で、各ベンダーはメインフレームの継続性とレガシーシステムの受け皿の考え方について、どの製品が有力であると考えているのだろうか。実際にメインフレームを製造販売しているベンダーの担当者にインタビューを行うことにより、各ベンダーのメインフレームの製品の継続性、オープン化した際の受け皿の技術、レガシーシステム移行（マイグレーション）の方針について調査を行った。

---

[15] 本図においては、全ての製品をラインナップしていない。また各ベンダー間の製品が横並びであることや、ベンダー内の製品の上下位置についてはあくまで目安であり、個々の製品の性能を決定付けるものではないということを留意していただきたい。なお、本調査・分析結果は、当該ベンダーの了解を得たものではない。

・ 日本電気株式会社（以降：NEC）

NEC は ACOS と呼ばれるメインフレームを製造、販売し、国内において多くの金融機関や官公庁への導入実績を持つベンダーである。レガシーマイグレーションの経験も多く有している。

ACOS 製品の継続性については、NEC は今後とも ACOS の開発、進化を続けていく方針であるという。既に ACOS-4 系である iPX9000 が Itanium2 を採用しているように、NEC 独自の高速化、高信頼性技術とオープンテクノロジを組み合わせ、今後とも継続的に後継製品をリリースしていくということである。また、オープン環境に移行する場合には、プラットフォームとして Itanium サーバー製品である NX7700i を主力と考えているようである。既に NX7700i は HP との協業によって多くのミッションクリティカルシステムで採用されていることが良く知られている。また、Itanium2 サーバーにおける稼動実績 No.1 を誇っているため、それに裏打ちされた安定したパフォーマンスがを期待される。また、ミッドレンジからローエンドをカバーする Xeon プロセッサが搭載された Express5800/スケーラブル HA サーバーにおいてもより一層の機能の充実化を図っており、商用 Linux をベースに障害原因特定の機能等を付加した高可用 Linux を用いることによって、x86 アーキテクチャのサーバーにおいても、高い可用性を必要とするアプリケーションに対応することを可能としている。

レガシーシステムの移行方法や基準については、基本的にはクライアントの要望を重視するという方針であるという。実際に、既存で安定して稼動している ACOS を所有するクライアントは、資産の継続や活用を望まれるケースが多い。その場合は ACOS 系を存続させる方針で臨み、全く新規で高可用性を求められるクライアントであれば、基本的には Itanium ベースのサーバーで構築する、という方針でレガシーマイグレーションに取り組んでいるようである。

・ 日本アイ・ビー・エム株式会社（以降：日本 IBM）

日本 IBM は System/360 と呼ばれた、今日のメインフレーム根幹を形作った製品の後継機種を販売しており、国内において多くの金融機関への導入実績を持つベンダーである。メインフレーム開発と提供に関しては長い歴史を持つ

グローバルカンパニーであるため、特にレガシーマイグレーションという定義はしていないものの、多くのメインフレーム移行経験を多く有している。

メインフレームの継続性という観点では、日本 IBM は他社とは異なるビジョンを提示している。IBM におけるサーバー製品は、ハイエンドである System z、POWER6 搭載サーバーPower Systems、x86 サーバーSystem x、そしてミッドレンジからローレンジまで幅広く対応する BladeCenter の四つに分けられる。従来の枠組みに当てはめれば、System z がメインフレームということになるが IBM は特にメインフレームとは明示せず、Linux とこれまでの IBM サーバーのプログラム資産が継承できる、IBM サーバー製品群におけるハイエンドサーバーという位置付けである（参照：補足 1）。概念的に他のサーバーと分けていないことで、オープン化かメインフレーム使用継続かの二者択一ではなく、System z を活用し、既存資産を活用ながら部分的なオープン移行やオープンアプリケーションの統合を促すという新たな提案を行うことにより、ユーザーに対してのレガシーマイグレーションの印象を一変させることに成功している。但し、この考え方がうまく合致しないユーザーが存在するのも事実もあるとのことである。

一方、レガシーシステムを移行する場合には、安易にオープン化するという判断をしないという観点から、そもそも「オープン化」とは何か、「オープン化」することのデメリットは何かに立ち返って考え、既存のメインフレームの良さを確認しながら部分的な移行を行うことが多いようである。

例えば、オープン化の技術進展は早く、3 年から 5 年でバージョンアップをベンダーから要求されることは稀有なことでない。この時、当該製品のみのバージョンアップで済めばよいが、諸要素に分割されているオープンプラットフォームでは、一つの製品のバージョンアップが他の製品にも波及し、結果的に多くのバージョンアップや改修費用がかかってしまうという事が良くある、ということである。実際に金融機関の事例においては、核になる継続的なアプリケーションはメインフレームで、環境変化が激しく、それに対応する必要のある顧客向けのアプリケーションはオープン技術でと住み分けがなされているとのことである。

・ 株式会社日立製作所（以降：日立）

日立は、元々HITAC（現在は用いていない呼称）と呼ばれたメインフレームを製造、販売し、国内において多くの社会システムでの導入実績を持つベンダーである。レガシーマイグレーションの事例についても多く公開している。

メインフレームの継続性については、2008 年 7 月に AP8800 と呼ばれる新製品を出荷し、単一筐体でレギュラータイプからハイエンドタイプまで 52 モデル、約 180 倍の性能幅をカバーするラインナップを提供している。このことから、一定の継続性があると判断できよう。また、以前の製品（AP8000）でサポートしていた Linux への対応を BladeSymphony へ移行することにより、オープンサーバーとしての機能は廃止し、一方で日立製品である統合システム構築基盤 Cosminexus との連携を強化することにより、システム全体をプレゼンテーション層、アプリケーション層、データ層の 3 階層化することを可能にし、オープンシステム環境における中核サーバーとして動作しながら、変化にスピーディーに対応できる基盤を提供している。

また、オープンシステム基盤としての受け皿には、BladeSymphony を用意している。この BladeSymphony は、サーバー、ストレージ、ネットワークを一体化したシステム製品であり、システム管理製品である JP1 を利用することにより、統合的でしかも柔軟に拡張可能なシステム基盤を提供可能としている。また、独自のハードウェア仮想化機構である Virtage（バタージュ）を有し、プロセッサやメモリ、I/O 等のハードウェアリソースのブロック化による論理サーバーの構築機能を有し、複数 OS の混在環境においても統合的な運用を実現している。

レガシーシステムの移行については、「レガシーシステム再生ソリューション」と呼ばれるサービスを提供している。但し、必ずしもレガシーシステムのオープン化のみを推奨しているということではなく、プログラムの移行性に着目した分析を行い、その結果を踏まえたソリューションの提案を行っている。具体的には、現有の資産をオープン言語にツール等で変換が行えるかどうかの棚卸しを行った結果を判断し、移行の難易度を分析している。例えば、難易度が高いと判断される場合には、移行に多くの費用がかかることとなるため、オープン化の目

的によってはメインフレームを使い続けることの方が妥当な選択たりうることとなる。

また、レガシーシステム再生ソリューションは以下で構成される。

- オープン化移行支援

  既存のプログラム資産を効率的に分析し、再利用可能な資産を有効活用することを前提に、システムの移行を支援する。移行計画～移行設計～オープン化移行～テスト支援までの一連のプロセスを踏むことにより、マイグレーションを実現する。

- 異種言語変換支援

  既存プログラム資産を可能な限り再利用し、ツールを駆使した高い機械変換率による高品質移行を行う。また、画面や DB 等を含めた 3 層アーキテクチャのプログラムソースに変換を行うことで、再利用性・保守性の高いマイグレーションが可能とする。

- C/S 系言語バージョンアップ支援

  既存のプログラム資産を各種言語の変換資料、過去のノウハウをもとにバージョンアップのための調査を行い、変換ルールを策定する。これまでに行った各 C/S 系言語の変換ルールを蓄積しており、それらのノウハウを利用しながら、バージョンアップ支援ワークベンチ（VWB ツール）による機械変換を行う。

・ 富士通株式会社（以降：富士通）

富士通は、元々FACOM（現在は用いていない呼称）と呼ばれたメインフレームを製造、販売し、国内において多くの金融機関、官公庁、社会システムの導入実績を持つベンダーであり、国産メインフレームを牽引しているリーダーである。レガシーマイグレーションの事例についても多く有している。

メインフレームの継続性について、富士通も 2009 年 5 月に GS21 と呼ばれる大型機にランクされる新製品を出荷し、処理能力を最大約 30%向上させるだけでなく、消費電力を最大約20%削減させている。このことからみても、メインフレームという製品の独自開発が未だ進んでいることを示していると判断できよう。また、当該製品では、安定性についても SSU（System Storage Unit） とクラスタ

をつなぐインタフェースの信頼性、可用性、保守性を強化したことにより、インタフェース障害時にも二重化構成での運用を維持することを可能とし、片系運用となっていた従来と比べ、更なる連続運転性を向上させることに成功している。

また、オープン化した際の受け皿となるサーバーには、戦略商品である、Itanium プロセッサベースの PRIMEQUEST がある。この PRIMEQUEST にはメインフレームで培われた高信頼性技術がハードウェアに活かされ、他ベンダーが展開する Itanium サーバー製品を超える機構を有していることから、オープン基盤を選択する際の第一選択肢となりうるかもしれない。

更に、標準ベースかつオープンソースの IT 開発を推進することを目的として、長期的な視野で The Linux Foundation に参画し、仕様の提案、標準化作業に貢献しており、これによりオープン化後の Linux 環境の長期サポートを約束している。

レガシーシステムからの移行に関しては、既存の COBOL 資産が変化の激しい業務において使用されているか、そうでないかを判断することにより、オープン化すべきか、メインフレームを使い続けるかを判断する、というスタンスを取っている。どの業界においても、変化の少ないコアと呼ぶべき業務と、変化の多い周辺業務が存在し、変化の多い周辺業務が、コア業務と連携しながらシステムを形作る形態が理想のアーキテクチャであるとの事である。また、変化の激しい部分には、順次 SOA を適用していくという解もあるという提案もあった。

加えて、富士通では Interstage ミドルウェア、NetCOBOL 運用パッケージ等を総称し、マイグレーションスイートとして提供している。

このマイグレーションスイートは以下の特徴を有している。

- 効率的なアプリケーション移行支援

  メインフレームと共通性の高いアプリケーション実行環境を提供することで、メインフレームのアプリケーション資産やデータ資産の流用性を高めると同時に、効率的な移行ツールを用いることで、短期間での移行を可能である。

- アプリケーションの堅牢な実行環境を提供

マイグレーションスイートはミドルウェアとして動作するので、変換ツールや言語ライブラリによる移行と異なり、トランザクション制御を備える等、メインフレームから移行したアプリケーションを稼働させるために相応な堅牢なアプリケーション実行環境を提供する。

- 高品質・高運用

  富士通がこれまでのホストシステムで培った OLTP 技術をベースに、ホスト資産の流用性を高めることにより、現在稼働しているアプリケーションの品質をオープンプラットフォームへ引き継ぐことが可能になる。また、オープンプラットフォームの運用機能を強化することにより、基幹業務や大量バッチ処理にも耐えうる高い運用性を実現する。

以上の様に、各社それぞれ考え方や方針は異なるものの、

① メインフレームの継続性は確保している
（今後もリリースが継続される見込み）
② メインフレームの受け皿となる高信頼性を備えたオープンサーバーを提供している
③ メインフレーム継続が、オープン化かという二者択一ではなく、ケースバイケースで検討する

との教示を実例と共に得ることができた。

更に、これら主要ベンダーのアーキテクト[16]は、性能や安定性の観点からは、「チャネルの能力や OS の安定性等では一部にまだ差があるものの、（これらのハイエンドーミッドレンジサーバー群は）メインフレームになんら引けを取らない」という意見も多く聞かれた。ただ、そのような状況だからと言って単純にオープン化ではなく、各社のレガシーマイグレーションサービスがそうであったように、段階的なオープン化や、メインフレームが混在した環境での運用も可能である。その様な構成も有用

[16] この報告書内での意義は「IT アーキテクト」であり、建築設計者ではなく、システムにおけるシステム全体の構成及びシステム間の結合について設計を行い、システム全体的な調和を担保する役割を負う役職を指している。

であるという判断に基づき、「オープン化か、メインフレームの継続か」の選択は、性能や安定性を見極めるというよりは、現状のアプリケーション特性（利用頻度や業務ボリューム等）やその業務の継続性、またオープン化したことによる頻繁なシステム変更（バージョンアップ等）の必要性とそのコストを見極めたうえで、適切なプラットフォームを選択することがより重要になっていると言えよう。この点については、3 章で詳しく述べる。

補足 1:Linux に対応したメインフレームの存在

メインフレームの用途は、主に独自仕様のアプリケーションに制限されていた。しかし、現在ではメインフレーム上で稼働する Linux 及びそれを稼動させるプラットフォームが成熟しつつある。このことから鑑みると、もはや昔の定義のメインフレームではなく、オープン環境における「ハイエンドサーバー」として扱うべきである。

事実として、メインフレーム用ハードウェアの高可用性と、より小さい消費電力と高い冷却要件を備えた IBMSystem z の様な Linux 稼動を保障しているサーバー上で、z/VM の極めて高い仮想化機能によって実現可能とした高度な統合レベルを実現したオープンソースアプリケーションの事例が増加している。また、今後もこの傾向が続くと予想される。

更に Linux 専用メインフレームを用意できるということは、これまで一度もメインフレームを利用したことがないユーザーにメインフレームを提供できる事例も現れる可能性があるということになる。すなわち、新しく、強力でかつキャパシティの大きい、ハイエンドのサーバー統合プラットフォームとしての用途の可能性があるということである。これにより、更なるオープンソースアプリケーション環境の成長と成熟が実現することとなり、メインフレーム環境を利用できる Web2.0 以降の、よりリッチなアプリケーションの新たなポートフォリオが拡大する可能性があることを示している。

今後、メインフレーム上の Linux はこの 5 年間に成長を続け、この環境をサポートするアプリケーションが増えるにつれて浸透率は 50%を超えるとガートナー社では予想している。大きな理由は、System z が元来 PC 向けに設計された OS にセキュリティ、可用性、極めて高い統合性を提供する独自の利点を備えていることである。その結果、System z は Linux が「データセンターに対応」した堅牢なものであるかのような印象を与えることに成功している。通常 x86 サーバー上に展開した基幹系 Linux の大半では、IT 組織が「メインフレームに類似したインフラストラクチャ」を x86 アーキテクチャ 上(例えば OracleRAC 等)で構築しなければならないため、こういった課題に直面しているユーザーは潜在的に System z のユーザーとなりえてしまうのである。

また、このメインフレーム Linux という現象は IBM に限ったものではないのかもしれない。事実、他社製メインフレームであっても、IBM 互換機であれば、基本的にハードウェアアーキテクチャについては同等のはずであり、理論上は Linux を動作させられるはずである。しかしながら、非互換の部分であったり、独自の I/O 関連装置等のドライバは別途用意する必要があるだろう。日立製メインフレームの AP8000 は Linux をサポートしていた、という情報を記述したが、おそらくこの方法によって Linux を実装していたと考えられる。日立は変化への対応の難しさという点から、この Linux のサポートをやめ、ミドルウェアとの連携という形でオープンシステムとの協調を模索しているが、他社ベンダーから今後、メインフレーム Linux という形でのオープンシステムとの協調や、ハイエンドの仮想サーバーとしてのコンセプトが出てくるかもしれない。

ガートナー社レポート「IBM メインフレーム（System z）のオープンソース化：2008 年」より一部加筆

2.1.2　新しい基幹システムの考え方

メインフレームの受け皿となる基幹系サーバーの選択肢は確実に広まってきており、それらを利用した移行も確実に進んできて、後述するように事例も積み重なってきている。

しかしながら、メインフレームにも保守性や運用性の良さがある点にも注意する必要がある。特にオープン系サーバーにおいて時折みられる不具合対応という理由でのパッチの導入や、サポート切れ等の理由でのバージョンアップ等の要求がなされないために、メインフレームでは動作環境やランニングコストに不確実な要素が少ないことや、長期的なサポートが受けられること等のメリットが見直されているのも事実である。

図6はガートナー社がメインフレームのオープン系システムへの移行状況について455社を対象に行った調査結果である。これによると、全てをオープン系に移行済み、または、移行予定に関して41.9%であるのに対して、方法は様々であるが何らかの形で「メインフレームを使い続ける」という選択を行うユーザーが 59.6%と多い。[17]冒頭に紹介した図4において、2006年以降継続傾向が強くなってきているということからみても、メリットが見直されている傾向が伺われる。

これはオープン化の進展の裏返しとして、オープン化のデメリットとメインフレームを使い続けるメリットについてユーザーが理解をし始めたから、とも言える。

[17] 本報告書の最後の業種データ等の詳細を参考とされたい。

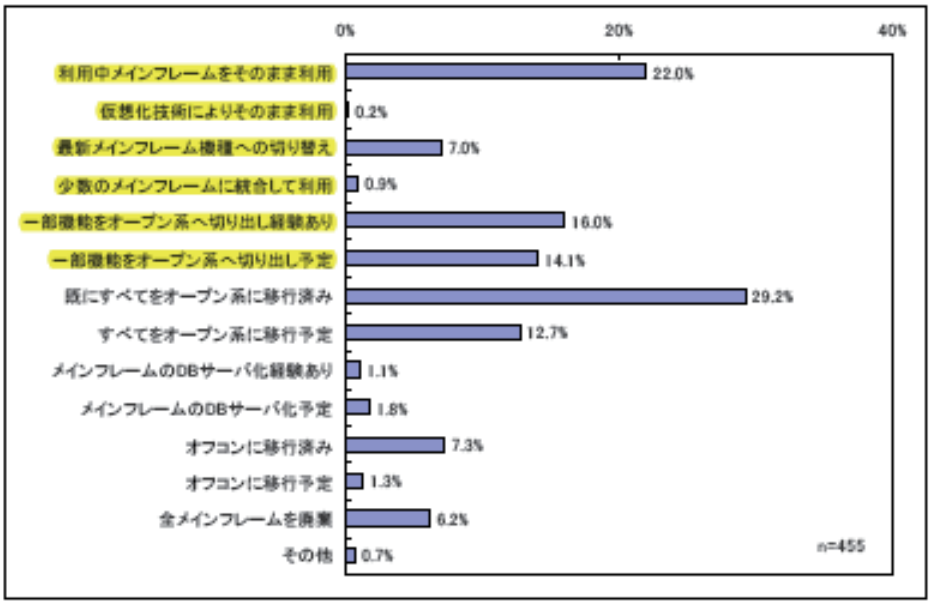


図 6　(複数選択) メインフレームのオープン系システムへの移行状況　2008 年 11 月ガートナー社調べ

オープン化のデメリットについては様々な観点があるが、技術的側面だけでなく運用の側面からも検討をする必要がある。オープン技術が企業に浸透するにつれ、その問題が明らかになってきたと思われる。特に、過度にオープン化を進めたために社内サーバーの分散化が進み、多数のサーバーが乱立したことによって、メインフレームと比べて負担が軽減されると言われてきた管理はかえって煩雑になり、社内のシステム管理者では手に負えないほどになっているケースが多いという意見があった。オープン技術を提供しているベンダーは「サーバーの仮想化」による統合という概念を生み出して、複数のサーバーをあたかも一つに統合したように管理できるソリューションを提供している。しかし元々仮想化という概念は、メインフレームが有していたもので、一つの筐体で様々なサービスを提供できるというアーキテクチャには一日の長がある。これが、ある種のメインフレーム回帰の契機ともなっている。

オープン化やメインフレーム使用継続の二者択一ではなく、多くベンダーが推奨していたように、業務要件に適合させながら、適材適所で混在させて利用する形態が基幹システムのアーキテクチャとしてはふさわしいという結論を得ることとなる。

こうした事実はさらに次の基幹系システムのあるべき姿について、一つの仮説を生み出すことになる。すなわち、メインフレームの良さ、オープンの良さに加え、新

たな要件への対応能力を兼ね備えた基幹システムのコンセプトを検討することが必要ではないかということである。

2005年からガートナー社が提唱している日本の基幹系システムに関するビジョンに「新基幹系システム」がある。これは、日本における基幹系システムに関する様々な混乱した議論、ベンダー戦略の不透明さを越えて、日本の基幹系システムを確立された前向きな議論へ持っていく必要性に対応するために生まれたコンセプトである。

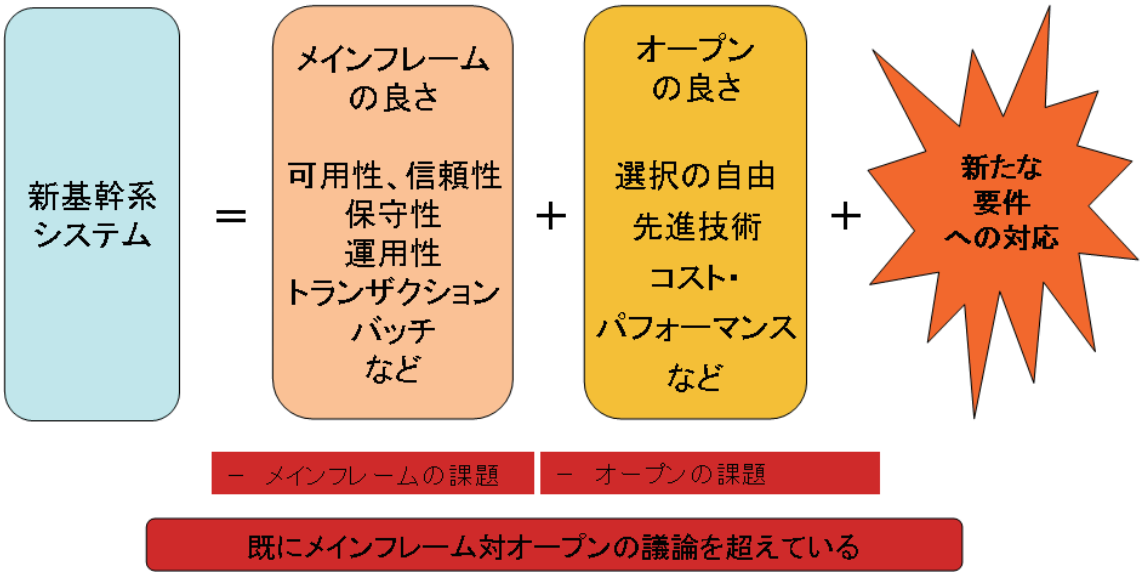


図 7　新基幹系システムの考え方（出典：ガートナー社）

新基幹系システムのフレームワークは、おおよそシンプルなものである（図 7）。基本的に、現在のオープンサーバーにおける様々な技術は、現場で新たにインタフェースを開発しながら等、摺り合わせの調整が必要になるのではなく、可能な限りベンダー間で最適化されたものとなる。

重要なのは、ベンダー間による技術の垂直統合を可能としている点である。この結果、これまでのオープンサーバーによる現場での調整、特に実績をベースとした摺り合わせを余儀なくされていた各種アプリケーションの実装も、ベンダーから出荷された時点である程度の完成度を持つものとなることが期待できる。例えば、現在

各所で議論されているサービス指向アーキテクチャ（SOA）[18]も、新基幹系システムの枠組みでは、ミッションクリティカルなものとして、提供している会社やその組み合わせに限らず実装できるようになるであろう。

また、メインフレームでは基本機能として組み込まれている、トランザクション、バッチ、大量印刷といった機能もシステム機能として明確化されるべきである。加えて、昨今オープンサーバーで注目を集めている、仮想化技術の実装はもとより、運用を可能な限り自動化する動きも新基幹系システムの枠組みで行われるようになるべきである。その際には、システム開発コストを削減するための、高い生産性の構築及び移行支援機能も重要となるであろう。

このように基幹システムの受け皿となるハードウェアが出揃ってきていることで、おぼろげながらも、新しい基幹システムに求められる要件が見えてきている。

[18] システムを構築する際の概念あるいは手法の一つで、業務上の一処理に相当するソフトウェアの機能をサービスと見立て、そのサービスをネットワーク上で連携させてシステムの全体を構築していく概念を指している。

補足 2:各メインフレームベンダーの将来構想

各社の将来構想についてもここで紹介する。

・NEC が考える将来像

NEC が考える将来像への移行はおおよそ以下の様なものである。

① 情報システムは歴史的にメインフレームによる集中処理からクライアントサーバーによる分散処理に変化してきた。

② 今後、統合型の集中処理(最適化)へと変化していく。

③ 利用形態としては、サービス提供型のクラウドコンピューティングへと進化していく。

④ サービス提供型へ変化するにあたり、大きく分けて 2 つのシステム群で構成する必要がある。

- データセンター設置に適した高性能/高効率でフレキシブルなサーバー
- システム全体のレスポンスや信頼性を補完し、現場設置向けにカスタマイズされたサーバー

・ 日本 IBM が考える将来像

日本 IBM は、日本企業は米国企業と比べ、効率化に関心が集中し、まだまだ戦略的な活用ができていないと捉え、今後、企業全体としての効率化や、IT を企業競争力に活用するために、ダイナミックインフラストラクチャと呼ばれるビジョンを示している。

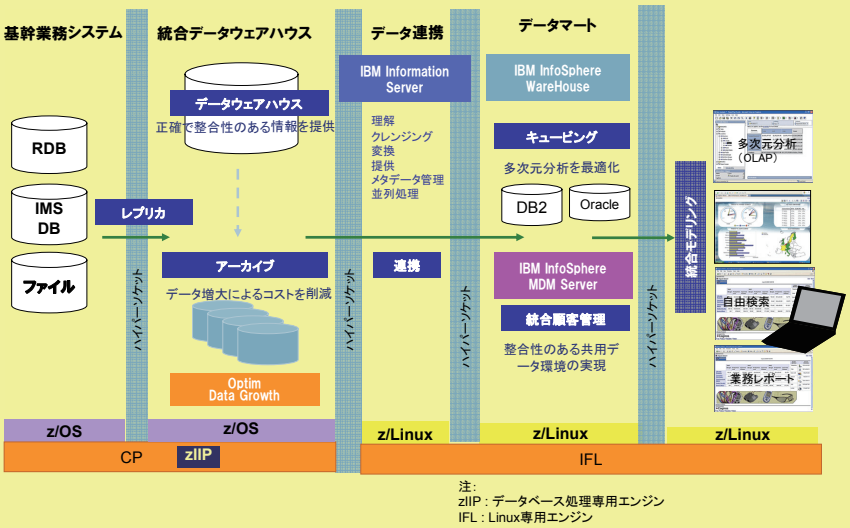
このダイナミックインフラストラクチャの要件を与えるために、より早く正確な経営 PDCA を回していかなければならず、以下の様な Performance Management が必要としている。

- Realtime – リアルタイムに情報を収集
- Unfiltered – 加工されていない生の情報を使う
- Shared – 全ての情報を誰もが共有できるようにする

➢ Holistic － 様々な観点から情報を収集・分析する

この PerformanceManagement の結果、必要なアプリケーションと必要な可用性等の非機能要件をまとめることにより、この要求に応じたインフラストラクチャを Blue Cloud という基盤を通じて動的に提供するというのが IBM のビジョンである。

この考え方は、現在の製品においてもある程度具現化され、z/Linux 上で稼動可能になった製品を組み合わせることにより、基幹システムのデータベースと統合・連携・最適化を通じて、いつでも必要なときに分散機を追加することなく、安価かつ高速に戦略的情報の活用の実現ソリューションを提供している。(以下参照)



参考:System z による Information On Demand のメリット

・ **日立が考える将来像**

日立は情報システムを戦略資源と活用するに当たって、まず複雑化した IT をシンプル化する必要があると考え、以下の三つの基盤が共鳴しながら価値を提供する、Harmonious Computing と呼ばれるビジョンを示している。

① コラボレーション基盤

- 業務システム間だけでなく、ビジネスパートナーのシステム、SaaS をはじめとする外部サービス等を柔軟に結びつけるコラボレーション基盤を実現する。

② エンタープライズ基盤

- 構成管理を大幅に自動化し、大規模な IT プラットフォームを容易に運用できるエンタープライズ基盤を実現する。

③ ユビキタス基盤

- ユビキタス社会の基盤としてセキュアで利便性の高い情報アクセス環境と、センサーや RFID をはじめとする多彩な手段を提供することにより、現場情報を効率的に収集し、柔軟に活用できるユビキタス基盤を実現する。

・ 富士通が考える将来像

富士通は、メインフレームは業務の中核として依然存在感を発揮し続け、また、多くのハードウェアにおいて、処理能力と信頼性を向上しながら、以下の様ないくつかの段階を経てクラウドコンピューティングへ向かっていくという将来像を示している。

① 分散化されたサーバーの集約化

業務内容や変化度合い等に応じて、ある程度の単位でのサーバー集約化が進展する。

② データセンター等へのサーバーの集約化

データセンター等で大型のサーバーやサーバー群に多くの業務アプリケーションが統合されて動作する。

③ 高度に統合されたシステム基盤（クラウド）

利用したいときに利用できる業務アプリケーションとそれに対応したキャパシティを動的に提供する。

2.1.3　高信頼技術の多様化

基幹系システムの信頼性を高めるためには、サーバーだけではなく、オープンシステム全体として信頼性を向上する技術が育ってきていることにも注目する必要がある。特に仮想化技術の進展に伴って、サーバークラスタ技術、データベースクラスタ技術、クラスタファイル技術、これら仮想化技術を包含しアロケーションを管理する高可用性ミドルウェア等、システムを構成するアーキテクチャ要素全般において、高信頼技術の成長が著しい。このような状況においてサーバー単体だけの信頼性を議論していても意味はなく、システム全体としての信頼性に着目するべきであろう。

メインフレームのハードウェアやソフトウェアが多くの実装実績があることもあり、耐障害という観点で優れた設計されている。一方、オープン環境における高信頼技術は、システムの中断を最小限に抑えることを目的に複数台のサーバーで「クラスタ」を構成しながら、一部のシステムを待機状態にしておき、故障もしくは高負荷となった作業をクラスタが組まれたシステム全体で担わせるようにするのが一般的である（図 8）。HA（High Availability）クラスタはこの方法を用いており、代表的なのは、ディスクシステムでの RAID である。RAID は複数台のハードディスクを組み合わせ（この状態をクラスタと称することもある）、仮想的な 1 台のハードディスクとして運用する技術で、ディスクアレイシステムにおける代表的な実装方法である。

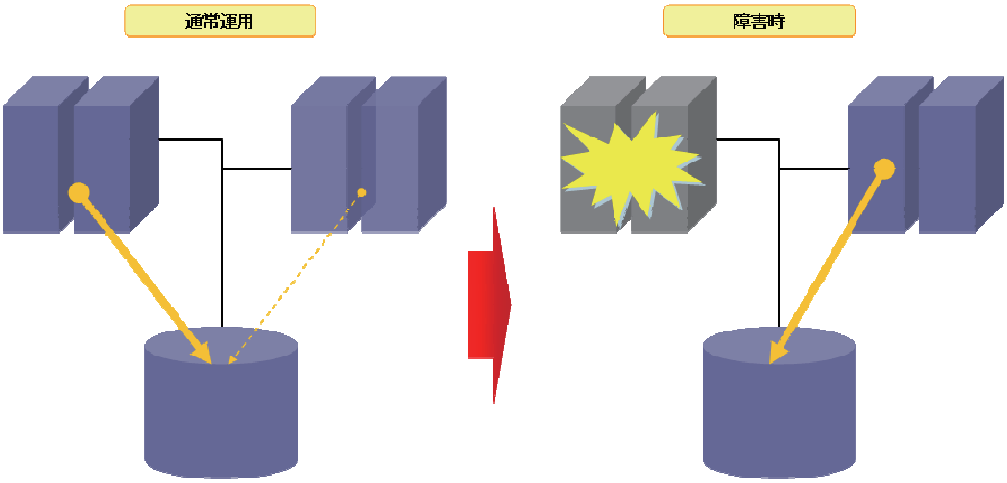


図 8　HA クラスタの概念

一方、ストレージ等においては、多くの大手ベンダーでスケールアップアーキテクチャをベースに信頼性の向上が進められてきた。この数年で、単一のファイルシステムイメージを提供できるスケールアウトアーキテクチャ（ノード［例えば、コントローラとディスクのセット］を並列的に加えていく）を提供するベンダーや方法が市場で浸透しつつある。

例えば、EMC の Celerra は N+1 のフェールオーバー機能を含むスケールアップアーキテクチャを採用し、従来の同製品よりも高い可用性を提供し、性能特性を予測可能としている。EMC と提携関係にある NEC のストレージ部門は、革新的な製品としてブロックベースでスケールアウトが可能な iStorageD シリーズ、バックアップ向けの iStorage HS シリーズの出荷を開始している。また、日本のストレージ市場ではリーダーとも言える日立も、Hitachi Essential NAS Platform を自社が強みとしているブロックベースの外付けディスクアレイ製品と共に販売することを強化している。

サーバー仮想化の波は、2 つの面でストレージ技術に大きな影響を及ぼそうとしている。

その 1 つは、VMware の仮想マシンの大規模導入に際して、複雑な VMFS（Virtual Machine File System）の管理を置き換える NFS（Network File System）を用いる方向性が強まっていることと、いくつかのストレージ製品が提供している「仮想ファイルサーバー」機能である。物理的に別々になった一般的なファイルサーバーを 2 台かそれ以上の拡張性の高い NAS ノードに統合し、ストレージレベルでリソースをプールするだけではなく、複数のファイルシステムを仮想的に統合する「グローバルネームスペース」の適用への関心も高まり始めている。

また、IBM はメインフレームを用い、別のアプローチで更なる信頼性を担保している。System z に実装される機能に、HyperSockets という高速な内部通信可能な機能があり、この HiperSocket を利用することで、ネットワーク接続（すべて内部）が簡素になり、TCP/IP スタックとメモリ間の移動の短縮により、セキュリティが高まるだけでなく、パフォーマンスが向上させることに成功している。特に、シェアードディスク方式で、ディスクをクラスタ内の全ノードにて共有し、障害発生時は障害ノードで実行されていた処理をそのまま他の生存ノードに引き継いで継続させる機構の Oracle RAC（Real Application Clusters）の実装では、ノード間通信を高速化できることから、RAC 全体のスケーラビリティを向上させ、安定したパフォーマンスを発揮することに成功している。

このように高信頼技術は 2003～4 年当時とは比べようもないほどに多様化し、充実してきている。これら高信頼技術は、必ずしもレガシーシステムからの移行を前提とした技術ではないが、メインフレームのオープン化においては、いずれかの技術を活用することによって、以前と変わらない高可用性システムの実現を強くサポートすることができるだろう。

2.1.4 システム統合の重要性

一方、高信頼技術と同時に、多くの組織や企業ではオープン技術の組み合わせに対するインターオペラビリティ（相互運用性）や柔軟性を求めるようになってきている。これは、オープン技術を組み合わせたり、摺り合わせたりしながら、信頼性のあるシステムを構築していくことの難しさやこれによって生じるユーザー側の「見えないコスト」の発生を抑える意味を包含している。

この摺り合せを実現するためには、単にプロジェクト、アプリケーションごとに最良の製品を選択して単に組み合わせたり、開発者 1 人当たりの生産性を向上させるようなアプローチでは実現しない。設計指針となる全体を俯瞰して、柔軟に構成が可能なアーキテクチャをデザインし活用することが重要になる。その有力なアーキテクチャを形作るものとして、SOA の認知度は一定のレベルに達しているが、現実に SOA を適用し、柔軟性という目的を達成できている企業は少ない。

そのような背景の下、SOA 等のアーキテクチャを実現する開発／稼働環境を提供するアプリケーションインフラストラクチャ製品は、複雑な要件に包括的に対応するために、アーキテクチャとテクノロジの革新を続け、新たなスイート化の方向性が現れている。しかし、その成熟度は進化の途上にあり、まだまだ改善の余地を残している状況である。

アプリケーションインフラストラクチャと SOA

アプリケーションインフラストラクチャは SOA バックプレーンとしての技術的役割を果たし、SOA に関連した相互運用性と統合を可能にするソフトウェア基盤である。WSDL（Web Service Description Language）、BPEL（Business Process Execution Language）といった標準もまた、アプリケーション開発を容易にする要素ではあるが、Web サービスによって相互運用性が保証されている。

SOA 基盤を設定するということは、ソフトウェア・プラットフォームに関して技術的な選択をすることを必要とし、アプリケーション・サーバー、ポータル、統合スイート、BPM ツール、トランザクション処理モニタ、統合サーバー、コミュニケーション・ミドルウェア、方針管理及び施行ツール、そして XML アプライアンスは、すべて大規模な SOA イニシアティブで役割を果たすものである。

ユーザーは、適切な選択をするために、ミドルウェアの複雑な世界を理解しなければならない。SOA の認知度が上がり、利用可能性が証明されているにもかかわらず、SOA を実現するミドルウェアは、新参者にとって間違った意思決定をすることによるリスクがのしかかってくる。

出典：Web2.0 や SOA の電子政府・自治体への活用に関する調査研究（平成 20 年 3 月　社団法人 行政情報システム研究所）

また、クラウド[19]や SaaS[20] 等の新しい概念やビジネスモデルが出現する中、これに呼応する形で、アプリケーションインフラストラクチャ製品も変革を遂げ、より多様なテクノロジが出現している。その結果、企業は最適なテクノロジ選択の意思決定を求められていると同時に、選択はますます難しい状況となっている。

その結果、最適化の実現に向けた新しいアプローチを採用してきた、歴史の長い企業における IT 環境は、複雑化／肥大化／冗長化の一途をたどっている。その

[19] クラウドコンピューティングはインターネット上にあるアプリケーションやリソース（プロセッシングや外部記憶等）をサービスとして利用する形を取り、利用料金を払うというモデルを指す。これまでのような、ハードウェア、ソフトウェア、データ等をベンダーの手を借りながら、自分自身で構築して、保有、管理しアプリケーションやソフトウェアを利用する場合と対比される。

[20] Software as a Service の略。アプリケーションソフトウェアをプロバイダ側のシステムで稼働させ、そのアプリケーションをネットワーク経由で使用し、そのサービスに対して対価を支払う形態と定義される。

ため、多くの企業で現状のシステム構成が把握できず、変更によってどのような影響が周囲のシステムに起こるか知りえない。このため、この変更のリスクがビジネスの阻害要因にもなっている。

一方で、ビジネスの環境変化は加速度的に速くなり、情報システムもそれに迅速に対応できる能力、すなわち「俊敏性」を求められている。モジュール化やミドルウェア等の活用は、確かに柔軟性を高めるだけでなく俊敏性にも貢献できるが、ビジネス環境の変化の速さに対応するためには、より大きな単位での情報システムの変化が必要とされる。いくら柔軟性を備えていても、機能単位での変更を必要な時間をかけて実装することでは、もはやユーザーの要求に応えることが難しくなってきたのである。

こうした状況に対し、ここ数年、システムスタックを構成する多くのレイヤ（例：ネットワーク、サーバー、ミドルウェア等)で統合や刷新による改善の取り組みが行われてきた。しかし、アプリケーションやソフトウェア基盤のレイヤでは、既存資産の再利用や共存も含め、最適化の実現に向けた進展が遅れていた。

2009 年 4 月 20 日、Oracle は Sun Microsystems（以下 Sun と省略）を買収すると発表した。本買収が完了すると、ソフトウェアとハードウェアの双方に大きな影響力を持ち、既存大手ベンダーと競合する新たな一大勢力が誕生することになる。

Oracle としては、2 社の持つビジネスアプリケーションからミドルウェア、データベース、OS、サーバー、ストレージ・プラットフォームに至る一連の製品セットを、「オープンでかつ統合されたシステム」の実現に生かすことができるようになる。これまで、Sun という企業の長期的な存続可能性に対する懸念が生じ、顧客の間には、他社製品への切り替えへのプレッシャーを含め、不安が広がっていたが、Oracle が Sun を買収することにより、Sun の事業継続性の確保は当面期待できることから、一先ず不安は和らぐことになる。

今回の Oracle による Sun の買収の背景には、「個別製品からシステム全体へ」という競争市場の大きな変化がある。すなわち、企業向け IT 市場は「必要なときに必要なサービス、リソース、情報」を提供できる能力を誰が有するか、というグローバルレベルの競争の時代に入っている。この事は、1章の図 2 においてユーザーの多く

が「オープン系技術力の不足」という課題を挙げているという事実に対して、ベンダー側からの一つの答えという意味合いを持つと考えられる。しかしながら、この事実は、オープン技術を選んだからといって、必ずしも分離調達が必要であるとは限らないという新たな課題を提起するだろう。

さらに、「2.1.2　新しい基幹システムの考え方」で述べたとおり、今後のシステム基盤には垂直的な統合性、たとえオープンなシステムであっても、個々のコンポーネントが単純に統合、連携できるだけではなく、個々の利用状況等も統合して管理しながら、可変的にリソースの割当やシステムの追加等が行える事が重要になってくる方向性に進んでいる。このことを踏まえたうえで、改めてレガシーシステムの移行を検討していくことがより重要となっていくだろう。

補足 3:アプリケーションインフラストラクチャに対する各社の取り組み

ソフトウェア市場におけるこれまでで最大の変化の 1 つは、ミドルウェア製品市場の出現であった。ミドルウェアは、その名が表すとおり、OS とアプリケーションソフトウェアの中間に位置する製品であり、最近までは特殊な製品として独特の役割を果たしていた。この 5 年間に、新たなビジネスアプリケーションの勢力拡大、ミドルウェアサプライヤーの製品開発、そして大手ベンダー買収、合併といった状況が重なり、この特殊な製品がスイートへと進化している。

また、先にあげた Oracle の例の様に、ここ 2 年間は、この分野全体に及ぶ買収によって、さらにその収束が加速している。

このアプリケーションインフラストラクチャには、ほとんどのランタイムミドルウェアだけでなく、次世代アプリケーション方式をサポートするアプリケーション開発ツールや管理ツールが含まれる。この次世代アプリケーション方式は、サービス指向アーキテクチャ (SOA)、イベント駆動型アーキテクチャ、ビジネスプロセス管理 (BPM) テクノロジに基づいている。従って、これらの製品、ソリューションはある一時点に一度に構築するものではなく、IT 部門におけるプロジェクトの 1 つあるいは複数のソフトウェアプロジェクトの基盤ととらえるべきであり、システムの統合をより助けるものとして位置づけられる。

こうした観点から、次世代のソリューション提供能力を推し量るうえで、また、複雑化したユーザーのオープンシステム構築をより簡便にするツールとして、アプリケーションインフラストラクチャへの各ベンダーの取り組み状況を研究することは、意義のあることだと考える。

- NEC

  NEC はバックエンドシステムのミッションクリティカルな機能の提供で長い実績があり、アプリケーションインフラストラクチャの研究開発への積極的な投資を行っている、特に、確かなサービス品質を備えながら、他社より迅速に標準を実装することに投資しており、Business Process Execution Language (BPEL) 2.0、Java Business、Integration (JBI)、Universal Description, Discovery and Integration (UDDI) 3.0 を世界的に最も早く実装した。また次世代 IT プラットフォームを「REAL

IT PLATFORM」と名づけ、柔軟で、安心できるインフラを快適に提供するとしている。

主要な製品は以下の通り:

- サービス実行基盤:WebOTX
- 情報管理:InfoFrame
- 開発環境:SystemDirector
- 運用管理:WebSAM

・ 日本 IBM

日本 IBM は幅広い製品群を有し、複雑で高性能なシステムサポートを提供してきた長い実績と、重要なアプリケーションインフラストラクチャ分野（ポータル、アプリケーション・サーバー、アプリケーション統合、データ統合、トランザクション処理、開発ツール等）における大規模なインストールベースを誇っている。とりわけ、システムにおいて単一ソースを求める企業に対する、製品とサービスの効果的な組み合わせを提供できることが強みである。

また、多くの標準規格活動や、SOA 中心の新たなコンセプト推進におけるリーダーシップをとっている。

主要な製品は以下の通り:

- WebSphere ソフトウェア群
  - ✧ アプリケーション・サーバー
  - ✧ エンタープライズサービスバス
  - ✧ ビジネスプロセス管理サーバー等
- Information Management ソフトウェア群
  - ✧ データベース＆データ管理製品
  - ✧ 情報統合＆ETL 製品

- ✧ ビジネス・インテリジェンス等

- Lotus
  - ✧ グループウェア製品等
- Rational
  - ✧ 統合ソフトウェア開発・管理ソフトウェア
- Tivoli
  - ✧ 運用管理ソフトウェア

・ **日立**

日立は従来型システム開発でアジア市場における高いプレゼンス誇っており、メインフレームとオープンシステムの混在環境の公共及び社会的ミッションクリティカルなシステムにおいて、広く構築・運用実績を誇っている。このことで、アプリケーションインテグレーションにおける市場においても、一定の存在感を得ることに成功している。

特に、統合システム運用管理分野においては他社製システムとの連携実績も多く有している。

主な製品は以下の通り：

- 統合システム構築基盤： Cosminexus

（Cosminexus シリーズは同社の基盤構築、連携ソフトウェアの総称であり、統合的なプロセス管理等の幾つかのコンポーネントが存在する）

- 統合システム運用管理： JP1
- 統合データ管理基盤： HiRDB

・ **富士通**

富士通は日本市場における主導的地位、日本以外の東アジア市場における高いプレゼンスを生かし、また、ハードウェアのブランド認知度とグローバルな勢力範囲を十分に発揮することで、このアプリケーションインテグレーション製品群の認知においても高いプレゼンスを得ている。

また、Interstage 製品群を補完し、システム全般にわたる統合性を確保するため他のベンダー（Software AG、IDS Scheer 等)と積極的に提携を進めることによって BPM における高いプレゼンスと、SOA 及び BPM の包括的な製品ラインを提供することに成功している。

主要な製品は以下の通り:

- ビジネスアプリケーション基盤: Interstage

  (アプリケーション&サービス管理と情報管理スイートがそれぞれ存在する)

- 統合運用管理ソフトウェア: Systemwaker
- 高信頼データベース製品群: Symfoware

2.1.5　業務及びシステムの特性に合致したプラットフォーム選択の必要性

以上の様に、レガシーシステムの受け皿となる基幹系オープンサーバーの選択肢は各社からハイエンドサーバーだけでなく、ミッドレンジの幅が広がってきている。また、システム全般として垂直的な統合管理を実現するビジョンに基づき、各社がシステム統合基盤に力をいれるとともに、ミドルウェア、データベース、ストレージ等に渡って高信頼技術が充実してきた。

しかしながら、根強いメインフレームへのニーズが存在していることで、一部で回帰の動きを見せ、各社からもメインフレームが継続的にリリースされている。加えて、新たなビジョンの提案や各社から将来的なシステム像が発表されている。これもまたレガシーシステムの移行先の選択肢となりうる。

これらのことから、基幹系システムの正しい刷新を実現するためには、単純にオープン高信頼技術を採用するだけでなく、メインフレームの継続も見据えることが必要である。それには、ユーザー自身のビジネス及びシステムの特性について正しく把握したうえで、全体的なシステムのアーキテクチャについて、刷新を契機に見直すことである。その事を通じて、システム全体としてより強力な機能を提供できる基盤に生まれ変わることを念頭に置くべきである。

そこで、次項以降では、システムの特性を機能面及び非機能面から業務要件から抽出し、適切なプラットフォームを選択することについての重要性と手法について検討する。

2.2　機能面から考えたオープン化移行方法の選択

2.2.1　移行すべきか維持すべきかの選択

前項において、業務及びシステムの特性から適切なプラットフォームを選択することが重要であること、また、システム全体としてのアーキテクチャを考えたうえで、適材適所な選択を行うことが重要であることを述べた。本項では、その適材適所な選択を行うためにどのような検討ステップが必要であるかについての分析を述べることとしたい。

レガシーシステムの移行においてまず検討するべきなのは、そもそも移行するべきなのかという点であり、そのための検証方法である。そもそもメインフレームで稼動してきたアプリケーションであるから、当該組織において重要なシステムであることはある程度想定できるが、時間の経過と共に、アプリケーションの重要性が相対的に低下していることもありえるし、逆に、時間の経過に関係なく、依然として基幹業務であり続けていることも少なくないと考えられる。そのようなアプリケーションについて、十分な検討なしに単純に「移行」という選択肢を選ぶことが、賢い選択ではない。

従って、現在稼働中、及び今後導入しようとするシステムにおける寿命、例えば、オープンシステムであればせいぜい 3 年程度の将来を考えて、業務機能の価値（当該組織においてどれだけの業務価値、すなわち業務的な貢献がどれほどあるか）と、そのレガシーシステム存続がもたらす、業務存続のリスク（当該組織において、レガシーシステムが存続することによる業務への影響度合い）の二点から評価を行い、図 9 に示したような枠組みのもとで大まかな移行方針を決定づけることが重要であると考える。

この図では、縦軸に業務存続のリスクの高低、横軸に業務機能の価値を取り、大きく四つの方針に分けて検討することを前提としている。

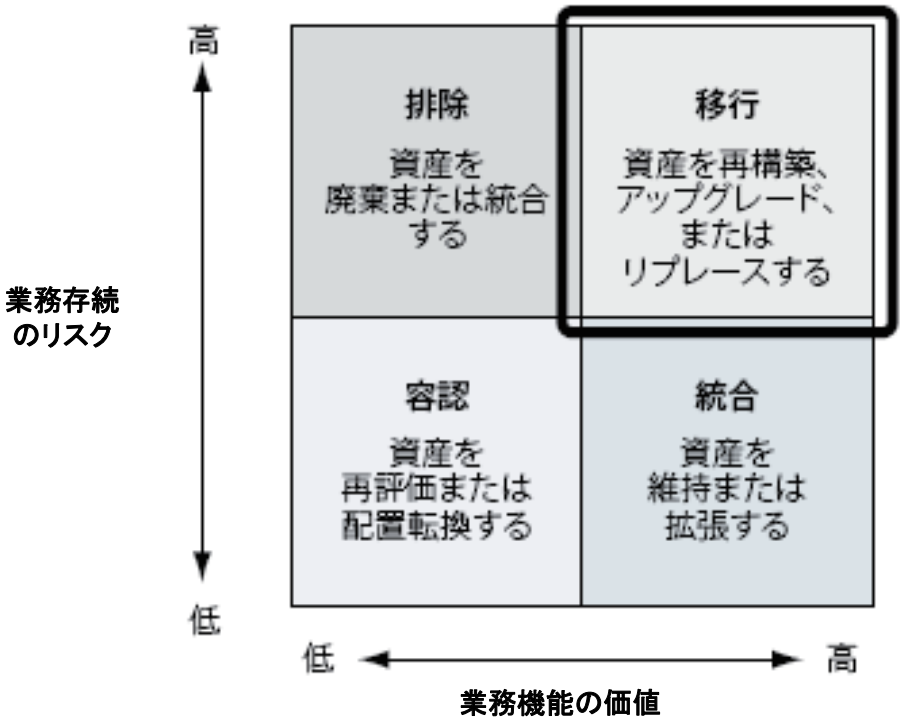


図 9　レガシーシステムからの移行判断①

この結果明らかになる各アプリケーションの価値とリスクのマップは、レガシーシステムからの移行の方針、計画すべき時期とその根拠を示すことになる。

業務存続のリスクが高く（存続することによって満足に業務がサポートできない可能性高い）、そもそもの業務機能の価値も高いと分析されれば、基本的な方針は「移行」となる。加えて、移行時期に関しても、なるべく早期に行わなければならない。

また、同様に業務のリスクは高いが、実際にはそのアプリケーションの業務機能の価値は低い、もしくは低下していると評価されれば、その稼動しているレガシーシステムは廃棄しても良い、という可能性が出てくるということになり「排除」という方針が立てられることになるだろう。

反対にレガシーシステム上の業務存続のリスクは低い（現行のレガシーシステムで運用することに何ら問題が無い）が、組織における業務機能の価値が高いという

評価であれば、業務が存続して問題ないという判断となる。この場合は、別のシステムと集約や、メインフレームの維持等で積極的に他のシステムとの連携を模索しつつ、システム全体としての機能価値を高める「統合」という方針が取れることになるだろう。

また、レガシーシステム上の業務存続のリスクも、業務機能の価値も低ければ、その業務の稼動を「容認」しながら、そのレガシーシステムとビジネス価値の再評価を行い、改めてシステムとして必要なものなのか、将来に渡って重要な業務となりえないのかを慎重に判断するということになる。

多くの場合、移行か継続か、もしくはオープン化かレガシー継続かという話に陥りがちであるが、重要なのは、移行戦略においても複数の選択肢を有しているということを認識することである。

上述のように、業務機能の価値が高く、業務存続のリスクが低い場合、単に継続するという選択だけでなく、新しいツールや技術を使用して機能を追加することにより、レガシーアプリケーションの耐用年数を伸ばすことができる。あるいは、中核のレガシー機能を最新のメインフレームにリプレースし、外部のオープンシステムと連携する形態とすることも可能である。これらは、レガシーアプリケーションの中核機能を使い続けるという戦術的なアプローチであり、システム全体でアーキテクチャを考え、先に紹介したアプリケーションインテグレーション基盤を駆使することによって実現できる。

いずれのアプローチの場合も、エンタープライズアーキテクチャに基づいて選択を導き、ハイリスクのレガシーシステムの新たな形成を回避することが必須条件となるだろう。

この考え方は、あくまで一例だが、昨今のレガシーマイグレーションにおいては、基本的なステップである「維持すべきか移行すべきか」の検討がされていないケースが多くあると考えている。事実、各ベンダーからも「オープン化への移行の目的が明確でない」等の意見が多く聞かれた。また、移行に関しては、必ずしも全て移行しなければならないということではなく、一部切り出してオープン化ということも考えられる。こういった判断を行うためには、業務という観点でリスクと価値を見極めることが重要でかつ基本的な検討のステップであると考えられよう。

2.2.2　移行方法の選択について

基本的な方針の検討の結果、レガシーシステム及びそのサポートする業務範囲を移行するという方針が固まり、更にその移行対象にメインフレームが含まれるならば、次に検討すべきはその移行方法である。移行の方法は、各ベンダーやシステムインテグレーターによって異なるが、一般的な移行の方法は大きく分けて以下の三通りであり、それぞれに特徴があり、図 10 のようなメリット及びデメリットを理解したうえで選択することが重要である。

移行方法	コスト			移行リスク	移行の主な目的
	移行	運用	業務効率		
リエンジニアリング	大	中 (長期的:小)	大	大	● 業務効率を改善したい ● システム機能拡充を実現したい ● 運用に柔軟性を持たせたい
リライト	中	中	なし	中	● COBOL資産を廃止したい ● オープン技術へ移行したい (要員等の問題で)
リホスト	小	小	なし	小	● 導入コストを低くしたい ● 階層型DBから脱却したい ● COBOL資産を活用したい

図 10　レガシーシステムからの移行判断②

リエンジニアリングは、ハードウェアから OS、ビジネスロジックに至るまで、全てを移行するという手法である。詳細にはパッケージ利用した開発とフルスクラッチと呼ばれる、ゼロから開発する方法に分かれる。メリットは完全オープンアーキテクチャに移行することで、ハードウェアやプラットフォームの選択に柔軟性が生まれることと、新しいビジネスロジックに対応できることである。デメリットは、移行コストが非常に高いことである。

リエンジニアリングの場合、既存の COBOL 資産の移行が困難であると判断され、また、COBOL 資産を棚卸しした結果、膨大な言語の変換費用がかかってしまうと判明し、ゼロからでもオープンでシステムを構築しなければならない事情が業務上の理由として存在するケースに採用されることが多い。また、その逆のケースで、そ

もそも多くの COBOL 資産を有しない組織が、棚卸しをするまでもなく、ゼロから新しくシステムを構築してしまおうと考える際に、採用されているケースも多く見られる。

ガートナー社の調査によると、メインフレームの移行を判断した企業の多くは、このリエンジニアリングを選択している。ある調査では、図 11 に見られるように 70%を超える企業がリエンジニアリングを選択していた（図 11）。これは、移行を決断した組織において、そもそもオープン化を前提とした移行が行われていることと、大きなコスト負担をするのであれば、業務ロジックを含めて改善を行うことによって、より大きな移行の効果を狙うという動きによるものであると考えられる。

リライトは、提供しているベンダーによっても内容や定義は異なるが、一般的には自動化ツールの活用を前提として、アプリケーションデータの構造を解析しながら、既存の COBOL 等の言語を、Java 等のオープン言語に書き換えることであり、そのメリットはハードウェアやプラットフォーム及びアプリケーションの選択に一定の柔軟性が生まれることである。ビジネスロジックに変更を加えないことで、要件定義や計画等の工数がリエンジニアリングと比べて削減されることから、移行コストについてはリエンジニアリングよりも比較的安価になる。ただし、既存プログラム資産の解析結果によっては、複雑な構造を手組みで解析し、修正することも必要になるため、思わぬ移行工数増加の可能性も孕んだ方法であると認識しておかなければならない。また、業務機能やプロセスについては、レガシーシステムのそれを引き継ぐために、業務効率等に対しての向上効果は見込むことができない。ただし、継続的で既に修練された業務において適用されているレガシーシステムであれば、元々リエンジニアリングを通して、業務プロセス等を改革する必要性は無いため、このようなケースにおいてはリライトがベストな選択であると言え、実際にそれが採用されているケースも少なくない。

リホストは、移行手法の中でもっとも変更範囲が狭く、比較的簡便でかつ低コストな移行手法である。一般的には、自動化ツールによって実行環境（バイナリ）の変更を行う。メリットは、低コストで階層化 DB 等から脱却し、ハードウェアやプラットフォームの選択に柔軟性を与えることであり、デメリットは、リライトと同様に古い業務プロセスをそのまま踏襲することや、古いコードの複雑性等の問題がそのまま残ることである。

以上のように、全ての手法には一長一短があり、移行目的によって移行手法は異なるはずである。よく言われる指針ではあるが、例えば、移行コストも含めた、短期的なコスト低減を目的とするならば「リホスト」が現実的であるし、既存の業務の見直しも含め移行したいということであれば、短期的なコストには目を瞑り「リエンジニアリング」を選択することにより、業務改善効果も含めてメリットを享受することを狙うことが望ましい。ただ、実際には必ずしもこのような合理的な判断が行われていないため、オープン化すればコストが下がるという信念の下、リエンジニアリングを行い、結果的に大きなコストメリットは得られていないというケースも多く存在しているのも事実である。しかしながら、今後は、上記のような判断を行うことで、より目的に合致したレガシーシステム移行を行い、より業務効率向上に貢献するシステムを構築して欲しいと考えるのである。

各ベンダーが実際にレガシーシステム移行を行う中で、プログラム資産の移行性という要素は非常に重要である、との回答が多かった。

前述のように、リホストもしくはリライトの移行容易性は、リエンジニアリングに比べると相対的には容易ではあると言われる。但しこの事は、プログラム資産の移行性にある程度の目処が立っていることが条件となる。あるベンダーの話では、移行方針が確定した後で、実際にプログラムの移行をツール等を用いて行ったところ、プログラムの複雑性が判明し、結果的に移行が中断した事例もあるそうである。

従って、これら移行方針を決定する際には、現有資産であるプログラムの内容を把握している組織であれば、その組織単独で正しい移行方式の判断が出来ると思われる。そうでない場合、ベンダーがプログラム資産について理解しているケースにおいては、ロックインされるかに関わらず、既存ベンダーの知識に頼ることで、確実な現有資産の棚卸しを行うことが望ましいと考える。すなわち、正しいレガシーシステム移行の判断を行うためには、現有の資産の棚卸しが不可欠である。そして、棚卸しは、一番理解している人、または組織が行うべきなのである。

正しい棚卸しの実施と、その結果は、プログラム資産の移行可能性を正しく判断する結果となる。それをもって、改めて業務上の変更の必要性が長期的にあるのか、ないのかという観点と、移行にかけられるコストを総合的に判断することによって、最終的に移行手法を決定することができる。そうすることによって、移行作業時に無

用な中断等が起こらず、円滑な移行作業や移行目的に合致した成果を得ることに繋がると考える。

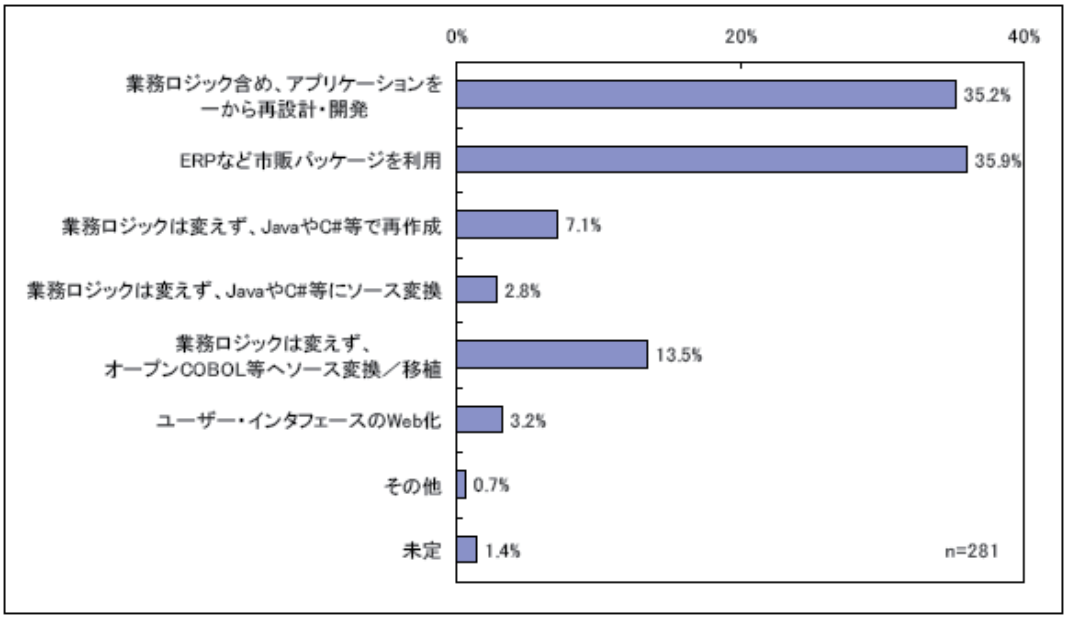


図 11　メインフレームをオープン系に移行する際の方法

2.2.3　システム特性という視点

移行方式と並行して、メインフレームにて現在どのような特性の業務を担っているのかについても、検討しておく必要がある。この特性については様々な定義があるが、一般的にはシステムにおける業務処理特性がどのようなものなのかというトランザクション[21]処理の観点と、データを単に参照するだけなのか、一貫性を保ちながら更新し続けるのかというデータ処理の観点によって定義づけられることが多い。

データ処理において参照を中心としたものは、参照系システムと呼ばれることもあり、単純なトランザクションではあるが同時並行的に多数を処理する必要がある。このような処理形態におけるパフォーマンスの向上には、いわゆる「スケールアウト

[21]　関連する複数の処理を一つの処理としてまとめたもの及びその単位。勘定系と呼ばれる金融機関のシステムにおける入出金処理が代表的であり、一連の作業を全体として一つの処理として管理するために用いている。

[22]」という拡張方法を取ることが一般的である。更新系処理においてもデータ分散や処理内容が単純である場合はスケールアウトによる対応が可能である。一方、後者は更新系システムと呼ばれ、パフォーマンスの向上には垂直統合的な拡張方法が取られ、「スケールアップ[23]」と呼ばれる。

アプリケーションの例で言えば、勘定系システムのような単一のイメージのデータベースに対して頻繁に更新が発生する、いわゆるオンライントランザクション処理システムでは「スケールアップ」構成をとることが多い。このことは、未だに多くの勘定系システムにおいて、メインフレームが使用される理由となっている。

2.2.4 システムの特性に合致したプラットフォーム選択

図12 においては、前項で取り上げた、ビジネス要件から導き出される、機能性能を勘案した際に「スケールアップ」が良いのか、それとも「スケールアウト」が良いのかによって、プラットフォームの選択肢が変わるということを具体的に示している。

大量かつ並列な処理でオンラインジョブとバッチジョブの混在という要件であれば、「スケールアップ」が向いている。「スケールアップ」であれば、垂直統合的なアーキテクチャを採用するプラットフォームが第一候補として挙げられる。この場合、依然としてメインフレームが有効な選択肢として浮上してくるということになる。もちろん、この判断は複合的であり、処理のボリューム等求められる要件や、業務プロセス等について変化の激しいものへの対応が要求されるようなケースでは、ハイエンドサーバーも有効な選択肢となりうる。この基盤選択の多様性については、2.1.1 で述べた通りである。具体的な選択肢については、本報告書の図5を参照しながら、各ベンダーから、構築実績とともに、情報を予め入手しておくべきであると考える。

---

[22] サーバーの数を増やすことで、サーバー群全体のパフォーマンスを向上させること。同じ OS やミドルウエアを持つ複数のサーバーで処理を分散することで、全体の処理能力を上げるというもの。比較的安価で処理能力を向上させることが可能であるが、多数のサーバーを運用する必要があるため、運用管理の手間が多くなってしまう面もある。Web サーバーなどで一般的に使用されている。

[23] 既存のサーバーを機能強化してパフォーマンスを向上させること。CPU やメモリなどの強化によってサーバーの能力を上げ、より高い負荷に耐えられるように拡張する。より強力な新しいサーバに交換することもある。比較的手間をかけずにパフォーマンス向上させることができるのが特長。

このように、どのような場合でも新しい高信頼性プラットフォームを選択するということではなく、システムの特性を鑑みながら適切なプラットフォームを選択することが、結果的に適切なキャパシティのサーバーを選択することに繋がり、無用なコスト負担を増やさないという観点でも非常に重要である。特に現在のアプリケーションがメインフレームでなければいけないのか、メインフレームを継続した方が良いのかについて着目するべきである。無理にオープン化するなどという、特定のプラットフォームに捉われるのではなく、業務特性に応じて、最適なプラットフォームを選択すすべきである。

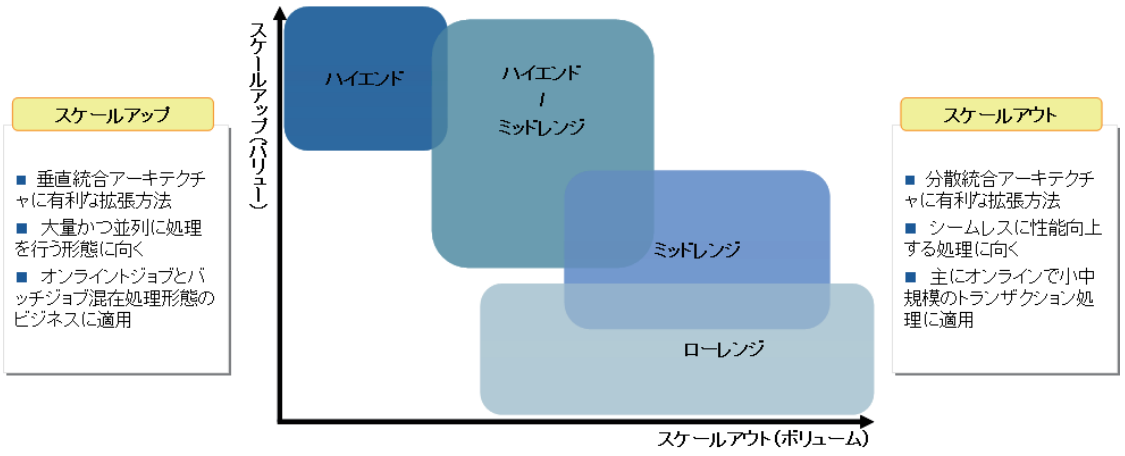


図 12　基幹系サーバーの選択肢とその基準例

2.3　非機能面から考えたオープン化方法の選択

2.3.1　非機能要件の設定とプラットフォーム選択

スケールアップ構成かスケールアウト構成かという大筋での機能要件が固まった段階で、どのカテゴリのサーバーを選択すればよいという判断まで行き着くことができる。次に詳細なシステム基盤の概要設計についても可能であれば行っておくべきである。実は、スケールアップ構成であるか、スケールアウト構成であるかについては、非機能要件に対する影響が大きい。

スケールアップ構成であれば、垂直統合型のハードウェア構成となるため、筐体単位での性能を高く要求する必要がある。一方、スケールアウト構成であれば、システム全体として性能を出すという思想になるため、筐体単体の性能要件はさほど厳しくなくなる。むしろ柔軟に構成を拡張していけばよいという考えになる。この前提を踏まえながら、非機能要件について可能な限り想定をしておくことが重要となる。

基本的に、調達側が提示する非機能要件はいつも曖昧で、「安全」「高信頼性」を要件として挙げる一方で、コストに対しての要求はいつも「低減」であり、調達の目的も「コスト削減」が大きいものを占めている。しかしながら、安全なシステムを構築しようとすれば、より堅牢にするための構成が必要であり、自ずと構築コストは上がりがちである。そこで、単純に「下げて欲しい」という相反する要求に対して応えるベンダーは、無理に単価を下げるしか選択肢が無く、苦慮しており、歪みを生じている。

そこで「どう安全なのか？」「どう柔軟なのか？」という問いに対して、一つずつ明確に答えることで、必要な非機能項目にコストをかけ、不必要な部分にはコストをかけない、といったようなメリハリを持った非機能性能を設定することで、コストと非機能性能を両立させたシステム構築に近づくと考える。

図 13 は、非機能要件レベル基準の一例である。参照系システムよりも更新系システム、組織内向けよりも組織外向けのシステムをそれぞれ重視し、段階的に非機

能要件を設定している。これにより、非機能要件における必要なレベルを定義している。本報告書の参考資料として全体を添付したので、参照されたい。

内容	選択肢	対象	Essential	High
システム特性	■基幹系システム □組織外向けWebシステム(参照系) □組織外向けWebシステム(更新系) □組織内向けWebシステム(参照系) □組織向けWebシステム(更新系) □C/Sシステム		■基幹系システム →可用性重視 □組織外向けWebシステム(参照系) □組織外向けWebシステム(更新系) →セキュリティ重視	□組織外向けWebシステム(参照系) →セキュリティ重視 □組織内向けWebシステム(更新系) →性能重視
利用対象ユーザー	■組織外不特定ユーザー □組織外特定ユーザー □組織内不特定ユーザー □組織内特定ユーザー	OS、プラットフォーム	■組織外不特定ユーザー	□組織外特定ユーザー
サービス稼働時間	■365日24時間(計画停止不可) □365日24時間(計画停止可) □8:00～24:00 □業務時間内	バックアップ	■365日24時間(計画停止不可)	□365日24時間(計画停止可)

図 13　非機能要件のレベル設定例（基盤選定テンプレートの一部）

このレベルは、非機能要件とされる項目においてそれぞれ設定されることによって、システムを実装する際の設計や要件の定義に役立つはずである。この項目はシステム特性、利用対象ユーザー、サービス稼働時間、障害からの復旧可能時間、参照処理のレスポンスタイム等に渡って、約 20 項目程度で設定されることが望ましい。

また、このような取り組みはベンダーやユーザー団体等でも盛んに行われており、特に「非機能要求グレード研究会（http://www.nttdata.co.jp/nfr-grade/）」等が有名である。各ベンダーの問題認識も本調査研究と基本的には同じで、不明確な非機能要求が、システム基盤部分のグレードに不明確さをもたらし、ハードウェア、OS やミドルウェアに加えて、その実装方式に影響を与えるという事実をこれまでの経験から強く認識し、その不明確さを積極的に除去しよう、という取り組みが行われている。

ただ、これらの取り組みには課題が多いことも事実であるし、万能の方法はないことも確かである。

このようなレベル設定における技術項目は、多すぎても担当者の過度の負担につながることになり、結果的に非効率さを生むことになるが、逆に少なすぎても要件を明確に定められないことになり、当初の狙いを達成できないことになってしまう。

そこで、例えば日本IBMは、業種によってアプリケーションをその特性ごとにグルーピングし、それごとにある程度の非機能要件を設定している。そのことで設計の質を向上させることに成功しているとのことである。

また、日立や富士通は、このような技術項目、技術要素は日進月歩であり、ユーザー側でレベルを設定したり、項目をアップデートすることの難しさを指摘したうえで、懇意なベンダーを複数作っておき、それらベンダーから定期的に情報を得ることで、過度なロックインを避けながら技術動向や選択肢の範囲等を把握していくことが重要であるとしている。

2.4　システム基盤の動向とその選択

2.4.1　移行時における技術基盤選択肢の多様化

今日、レガシーシステム移行を検討するときに、2003 年当時とは比較にならないほど、技術基盤の選択肢は広がっていることが確認できた。これはオープン系技術に限ったことではなく、メインフレームでも同様である。オープン系においては、高性能 CPU 開発が相次ぐことで、強力なハイエンドサーバーが生まれてきている。また、メインフレームにおいては、Linux が動作することで、オープン系のアプリケーションの動作基盤、統合基盤としても動作する等、もともと間違った認識ではあったが「メインフレーム＝レガシー」等という、これまでの概念が全く通用しなくなっている。

また、高機能、高性能となったのは、何もサーバーだけにとどまらず、データベース、ミドルウェア、ストレージ等、システムを構成する全ての要素において、高信頼性を獲得するに至っている。この全ての要素において高信頼性を獲得するという動きは、システム全体としてリソースを共有しながら、動的に配分するという構想につながり、多くのシステムベンダーにおいて、個々の要素を提供するというより、より垂直統合的なシステムを提供するという構想に変化しつつある。このことによって、オープン技術を組み合わせて利用するユーザーの管理負荷は軽減することになる。

今後における政府情報システムの刷新は、単なるオープン化ではなく、メインフレームの存続という選択肢を踏まえながら、システム全体で連携し合いながら機能を実現するという思想に変化してきている。その時に重要なのは、ユーザーがシステムを用いて、何を実現したいかという目的を明確にすることである。そのビジョンに応じてシステムを柔軟に組み合わせて、利用する時代がすぐそこまでやってきているということである。

2.4.2　機能面から考えたオープン化移行方法の選択

基幹系サーバーやそれを支える技術の基盤の選択肢が広がったことで、選択の良し悪しがレガシーシステム移行の大きな成功条件となった。

重要でかつ、省いていけないのは、「そもそも移行すべきかのか？」という問いについて、真剣に検討することである。言い換えれば、レガシーシステムと一括りにされているシステムは現時点において本当に「レガシー（過去の遺物）」なのかということについて、改めて考えてみることが必要である。このことを検証するために、存続リスクとビジネス価値を評価し、それによって大まかな移行方針、移行、継続、廃棄、再評価等を定めたうえで、移行方法を検討するというステップを紹介した。これを一連のフローとして示したものが図 14 である。

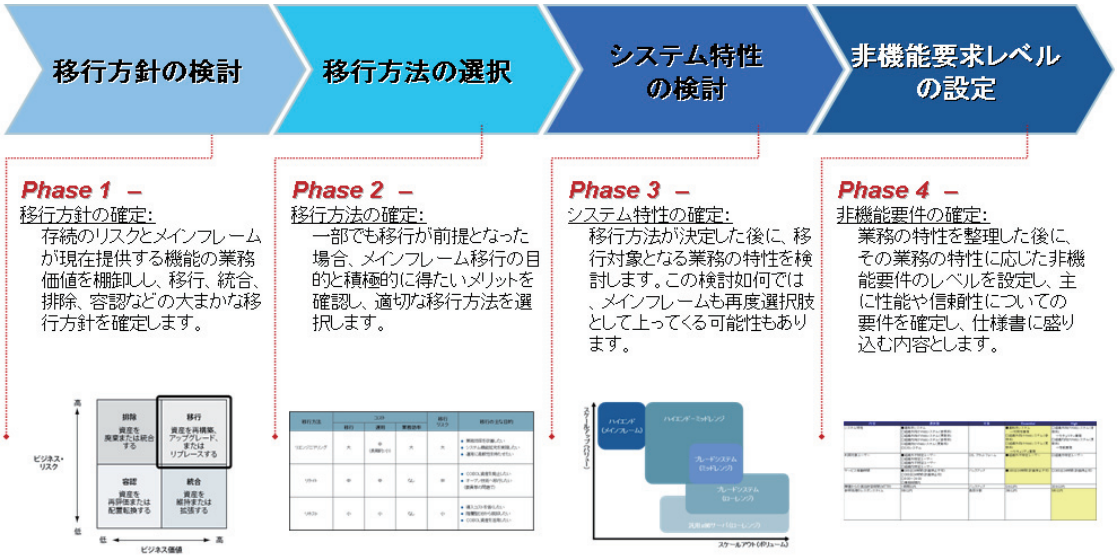


図 14　移行検討のステップ全体像

移行方法を検討するステップでは、市場にあるレガシーシステム移行方法を紹介しながら、それぞれの特色を勘案し、ビジネス上の目的、例えばコスト削減や、ビジネスプロセスの改善等に応じて、方式を選択すべきだとした。更に言えば、既存のプログラム資産の棚卸しをしておくことが、最終的に方式を決定する際に重要な要素になるということも付け加えた。そのうえで、システム特性と業務の特性を検討し、スケールアップかスケールアウトなのかという傾向をつかみ、移行先のプラットフォームを検討するというステップを紹介した。

2.4.3　非機能面から考えたオープン化方法の選択

機能要件によって、移行方針やシステムの拡張方針（スケールアップ、スケールアウト）が決定されたところで、非機能要件をどう定めていくか、についても検討した。

非機能要件は簡単に決められるものではないが、JUAS（日本情報システムユーザー協会）や非機能要求グレード研究会と同様の取り組みである、非機能要件レベルを設定する取り組みについて紹介した。理想的には、独力でビジネス要件から非機能要件に落とし込み、仕様書を作成するのがベストであるが、人材育成等の問題もあり一朝一夕には難しい。従って、アーキテクチャ構成基準を作成しながら、その基準を反映できるようなテンプレート（ドキュメント集）を開発し、その共有・活用によって、ビジネス要件に合致した、必要な非機能要件を達成するためのアーキテクチャを構成できるようになることが望ましいと考える。

以上の様に、システム刷新を取り巻く技術及びアーキテクチャ動向、そしてそのアーキテクチャを構成していくための検討プロセスについて研究を行った。その事を通じて、レガシーシステムの移行の受け皿が、単なるオープン化ではなく、メインフレームの存続を含めいかに活用するのかという点や、オープン化の方法も技術的に多様化してきていることを確認した。そして、それらをどのように選択していくかの基準を検討してきた。

次章では、実際の移行事例を参考にしながら、移行を行うにあたって、成功と呼べる条件はどのような点であるか、また、どのようなガバナンスが効果的であるかについて論じてみたい。

3. システム刷新におけるガバナンス

3.1 レガシーシステム移行事例とガバナンス

2章では、基幹系におけるシステム基盤は多様化していることから、レガシーシステム移行においては目的を明確にし、その目的に応じたシステム基盤の選択と移行方法を選定することが重要であることを述べた。この章では、各ベンダーの事例を用いて 2 章における提言の有効性を検証しつつ、更にシステム刷新に必要なガバナンスについて検討する。

3.1.1 NEC の事例

システム移行について、機能追加や開発の柔軟性の確保を目的とした事例を紹介する。

ある金融機関では、約 25 年間稼動させてきたメインフレームを UNIX サーバーと Oracle データベースで構成するオープン系の勘定システムへと移行した。その目的は、散在するデータベースの連携を図ることで、顧客情報の一元管理を実現することやアプリケーションソフトウェアを部品化し、柔軟な機能追加の機能を実現するとともに、長期的なコスト削減を副次的に実現することである。

また、オープン環境への移行によって得られると想定したメリットは、ライフサイクルコストの観点から長期的にコストを削減することだけでなく、UNIX サーバーが拡張性に優れていている点にも着目していた。金融機関においては、他の業種と比べて事業分野や業態の変動が大きくなく、今後大きな仕事量の増加を見込むことは現実的では無い。それでもサービスの多様化等は十分予想できることであり、その際に負荷に応じて増設が可能なスケールアウト的な発想を適用できると考えたからである。

運用コストを抑制しつつ、質の高いシステムを維持し、ユーザニーズに迅速に対応することによって、タイムリーな新商品開発を行うためのシステム基盤を実現する

ことを目的とし、アプリケーションソフトにはオープンアーキテクチャのパッケージソフトを採用した。そのうえで、NEC との緊密な連携により開発作業を進行した。プログラムはオブジェクト指向技術を全面的に採用し、商品毎単位に部品化した独立構造とした。また、プラットフォームには、ハイエンド Unix サーバーと Oracle のクラスタ構成を選択した。

しかし、構築に当たっては入念な準備を必要とした。例えば、当初予定していた稼動時期から 2 度の延期を経た。但し、この延期はトラブルが起きないように慎重を期したからである。当時、他の銀行のシステム切り替え等においてシステム障害が相次いでいたため、万全を期すことが求められていたという事情もある。例えば、残高照合やデータベースの整合性をチェックする機能等を盛り込んだうえで、テストを繰り返したという。

また、この金融機関のシステム部門の行員は、元々プログラムを書いてきたという経験があり、システムに関しての知識と設計のノウハウを蓄積することが出来ていたため、NEC に任せきりにせず、効率よく協働して構築を進めることができたことも大きかったようである。

かくしてシステムは稼動し、アプリケーションを部品化することによって、新しい商品の追加による機能変更は、約 2 割の部品改修で済むようになり、優れた拡張性と柔軟性を実現した。また、改修時に制御プログラムを追加することが必要になるが、この追加によってシステム全体には影響を及ぼさない設計となっている。この機能改修や追加の柔軟性により、長期的に見たシステム改修やメンテナンスのコストは低減する見通しが立っているようである。

この事例からわかることは、レガシーシステムの移行、特にリエンジニアリング[24]という方法は最もリスクが高く、最もコストの掛かる移行方法となることである。そのため、柔軟性という目標を定めながら、コストは副次的な目的とすること、そして新機能や新技術ではなく、目的とした柔軟性を有したアーキテクチャ構築を重視するこ

[24] 本事例の方式（リエンジニアリング）を判断するに当たって、関係者（開発メンバーや当該金融機関）に意思を確認はしていない。あくまで事例の内容から判断し、リエンジニアリングと分類付けている。

とが必要である。またユーザーとベンダーとのコミュニケーションを密にし、、多くのテストを重ねていくことは時間を要するが、最終的には成功への近道と言える。

3.1.2 日本 IBM の事例

日本 IBM の事例では、金融機関におけるオープン化を中心とした情報システムの刷新であっても、仮想化等によるサーバー統合や、Linux 化という流れの中で、アーキテクチャを検討し、最適なシステム構成を選択しているということだ。一般的な第三次オンラインシステム等の基幹系システムの置き換えをメインフレームで進めながら、周辺の顧客中心システムに関してはオープン系により構築し、システム全体として連携するというアーキテクチャで、支店システムの最適化を行っているケースもある。

例えば System z の導入事例としては、図 15 のように、他社メインフレーム(IBM 互換機)からの COBOL 資産の置き換えを行いながら、新たにメインフレーム Linux 上で構成した Oracle RAC で DB サーバー機能を担い、一方周辺系はブレードサーバーでアプリケーション・サーバーを連携させる形でアーキテクチャを構築して稼動しているケースがある。

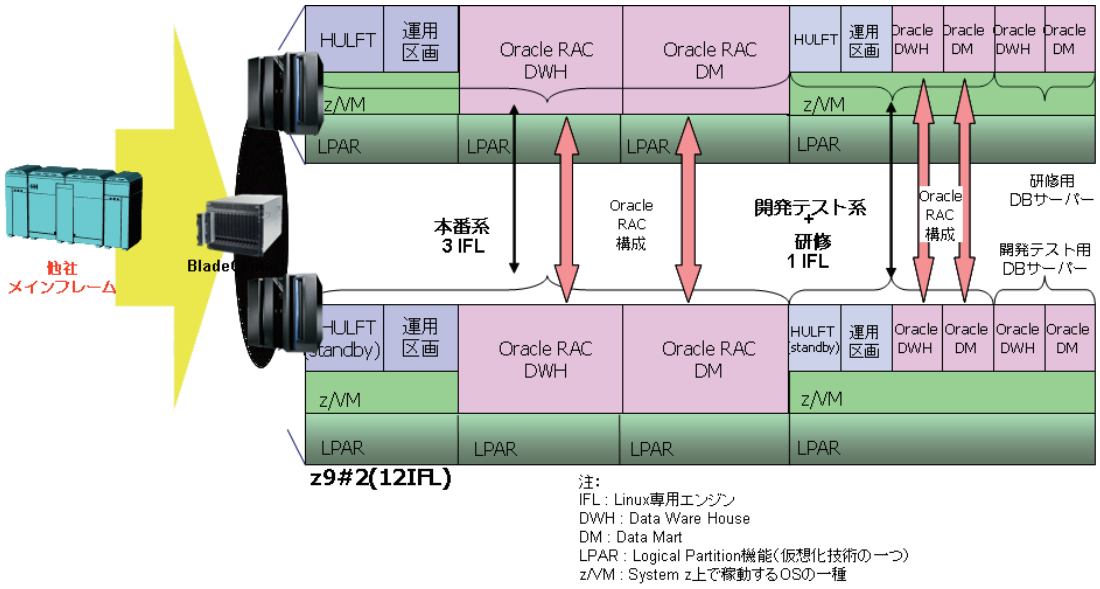


図 15 System z の導入事例(日本 IBM から提供された図表を一部修正)

これは、オープン化によるメリット、デメリットを改めて考えたうえで長期的、総合的にシステムを構築すべきだという考え方から来るものである。オープンであっても、レガシー（メインフレーム）であっても、基幹系に求められる信頼性要件は変わらないし、周辺のシステムで求められる要件も変わらないはずである。それであれば、時間的な業務の変化に応じて、適材適所のアーキテクチャを作るべきとしている。

一方、もともとメインフレーム以外のプラットフォームで動いていたコミュニケーション用途である、LotusNotes/Domino を System z（メインフレーム）で稼動させるという事例もある。

ある公共サービス企業においては、24 時間 365 日でサービスを提供しなければならないという責務を負っている。それは災害時においても同様であり、たとえ Notes のようなメールやコミュニケーションのシステムにおいても、全社に情報共有や交換のシステム基盤として浸透した後では、相当のミッションクリティカル性が求められていたとのことだ。この事例では、従来数十台の UNIX サーバーで運用されていた Notes を一台の System z で再構築し、更に 130km 離れたバックアップセンターの予備系一台とクラスタ機能により同期するという構成で構築しながら、メインフレームの統合管理によってアクセス制限を行うことで、強固なセキュリティを実現している。

このサーバー統合では、System z による仮想化技術を活用することで、新しくサーバーを購入することなく、論理的にサーバーを追加できる。またサーバー設置スペースやネットワーク機器の削減等、物理的な運用管理も含めて非常にシンプルになった。更に、統合されたサーバーにおいて高度なセキュリティ対策を実現したことにより、ログの取得や情報漏えいに関する追跡管理ができる仕組みが実現することとなった。

この二つの事例から言えることは、エンタープライズサーバーがメインフレームかオープンかという選択ではなく、要件に応じつつアーキテクチャを踏まえた全体最適の構成を考えるということが重要だということである。また、メインフレーム自体も進化をしており、メインフレームすなわちレガシーシステムというわけでもない。情報システムの形態は分散と集中を繰り返しており、IBM のメインフレームは古くから集中・統合の位置にいた。近年の分散化から再統合という流れにおいて、過度に分散化されすぎたサーバーを、System z のような堅牢でかつ高い可用性を持つサーバーに集中化／統合化を図るというトレンドが存在していることが事例から確認できたと言える。

3.1.3 日立の事例

日立の事例では、総合商社におけるミドルウェアを活用した、レガシーシステムからの移行の事例を取り上げる。この事例ではメインフレーム MP5600 で稼働していた財務会計システムの移行を計画するにあたって、オープンミドルウェア製品群を活用して、オープンプラットフォームへ完全移行することを試みた。

元々、この総合商社では SAP R/3 をベースとした基幹システムを国内本支社、海外現地法人、国内外事業会社に展開してきたが、最後まで残っていたのが外国為替や入出金決済等の商社における主力業務を担っていたメインフレームの MP5600 で稼働していた財務系の基幹システムだった。

外国為替や入出金の業務においては、大きな変化が少なく現行の機能面はもちろん性能面でも大きな不満はなかった。しかし、既に構築されているオープンシステムと比べて、運用費用がかかっているという事実に加えて、独自のプログラム言語で構築されたプログラムを多く抱えていたため、このプログラムを扱える技術者の減少により運用が困難になってきていた。それらの対応とともに、SAP R/3 との連携を実現することで、高機能化と変化への対応が可能な柔軟なプラットフォームへの移行へと舵を切ることになった。

移行する方針が決まったあとは、具体的にどのような方法で移行を行うかということになる。まずは既存プログラム資産を精査し、必要なものとそうでないものに整理をすることが、移行作業を効率化し、その後の保守性も高めるという考えの下に、既存プログラム資産の棚卸を行った。これにより、老朽化したものや、制度の変更等で使用されていなかったものが発見され、約 2 割のプログラムを削減できたという。

次に段階的なオープン化移行ソリューションを行った。2章で紹介した手法のうち、リビルドに類するものである。

旧メインフレームで使用されていた独自のプログラム言語を標準的な COBOL へ変換することで、先ずオープン化可能な洗練されたプログラム資産とし、続いて、移行資産に変換プログラムを適用し、これまで使われてきた実績のある業務ロジック

を極力修正しない形で COBOL2002 へと変換した。更に、並行して、ホスト上の XMAP2 を Windows 画面の XMAP3 へ変換、データベースも ADM から HiRDB へフォーマット変換し、更に JCL やジョブネットについてもツール変換や再定義によって移行することができた。

これにより、新財務システムでは、OpenTP1 と HiRDB の組み合わせによる信頼性の高い DB/DC 機能の提供等もあわせて、メインフレーム時代と同様の画面操作性を、日立の UNIX サーバーと連動した Windows 画面で実現した。 また、日立製ミドルウェアの適用により、将来を見据えた高信頼の Java[25]環境も実現しながら、全世界に 250～300 万人近くいるとされている COBOL プログラマーの経験やスキルを、今後もオープンシステム上での新規プログラムの開発や保守に活かせるフレキシブルな環境へと移行することに成功した。これにより業務ノウハウの継承と、低コストで柔軟性の高いシステム基盤の構築をともに実現できた事例となった。

この事例から言えることは、一気にしかも短期的にレガシーシステム移行を行うのではなく、綿密な計画の下、変えない部分は極力保存しつつ、変更する部分は慎重に変更を行うことが重要だということである。更にこの事例においても、ユーザーは決して丸投げせず、時間をかけて、変えない部分と変える部分について、その必要性を含めてベンダーと良く話し合っているとのことであった。

3.1.4 富士通の事例

富士通の事例においても、昨今のメインフレームの移行もしくはリプレースについては、オープン化かメインフレーム継続かという二者択一でなく、システムが担う機能によって使い分けるというアプリケーション資産をどのように最適化していくか、という傾向となっていることが分かった。

例えば、金融機関における移行事例を見ると、図 16 のように中核業務となる「預金/為替系」を担う系統は、長期的に確立されてきた安定的なサービスを提供しているため、プロセスへの変化が業務に起こることが少ない。さらに大量のトランザク

[25] OS やブラウザのようなプラットフォームに依存しないアプリケーションソフトウェアの開発を可能とするオブジェクト指向言語で、サンマイクロシステムズによって開発された、事実上の標準、そして仕様が開示されたオープン言語である。

ションを処理する能力が必要とされ、社会的な基盤としての安定性が第一に求められることから、メインフレーム上で稼動することとしている。他方、各種情報系のシステムや投信、窓口サポートシステム等は業務内容の追加や変化が激しく、更に複合的な商品提供の対応が求められるため、柔軟に変更可能なオープン系の技術で実装されることが多いという。

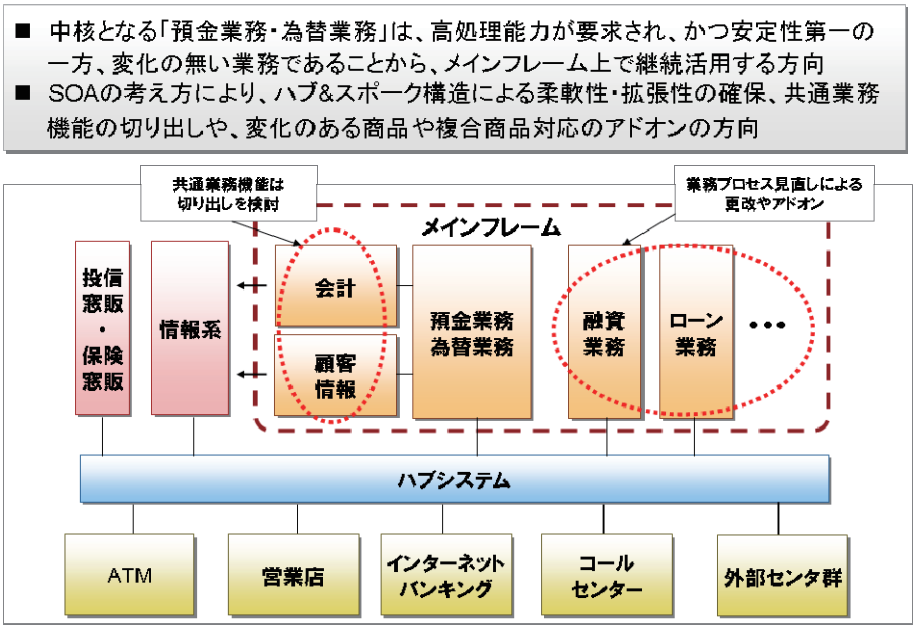


図 16 金融機関における移行事例(提供:富士通)

一方、ビジネス要件の変化がダイナミックな、多品種、大量生産、多頻度の商品更改等という業界、例えば一部の製造業や流通業、食品業界等においては、ERP等を用いたビックバンアプローチ[26]でのオープンシステムへの更改や、段階的にCOBOL 資産を UNIX プラットフォームで稼動するように移行していくという、大きく二つのケースがあるという。

また、導入時にはハードウェア資産としての単純なコストの比較は可能で、オープン化を行うことで多少の低減を実現することができるが、オープン技術のバージョ

[26] ERP 等の基幹業務システム導入において、各業務モジュールを段階的に導入するのではなく、必要な機能をすべて同時に一括して導入し、稼動させる方法。プロジェクトが大規模化、複雑化するため、導入失敗リスクは大きくなるが、業務面を含めて刷新効果が大きく出るため、成功した場合における効果は大きいものがある。２章におけるレガシーシステム移行方法ではリエンジニアリングに類する。

ンアップ等のコストは想定することが難しく、実際に長期的な観点で ROI[27]を調査してみないとコスト削減につながったかどうかは分からないため、富士通では長期にわたって ROI を計測するケースが存在するとのことである。

いずれにしても、単に「オープン化」を目的とするよりは、事業モデルの改革なのか、BI[28]等の情報活用の高度化を踏まえた改革なのか等のビジネス要件に対する認識をまず確認する必要がある。そのうえで、ビジネス環境の変化が激しいのかそれとも緩やかなのか、移行にかけるコストとリターンは適切であるのか、長期的なコスト削減を目標とするのか、部分的もしくは段階的な更改なのかをはじめとして、レガシーシステム移行にあたっての目的やゴール等をユーザーと明確にしながら進めていくことが重要であるとしている。

3.1.5 事例からわかること

ここまで各社におけるレガシーシステム移行の事例について紹介した。これら事例から分かることは以下の通りである。

1. **レガシーシステム移行の選択肢はオープン化だけではない。**

   レガシーシステムからの移行はオープン化が全てではなく、メインフレームの存続、必要に応じてメインフレームをアップグレードするという選択もある。日本 IBM や富士通によると、長期的かつ継続的に業務変化が少なく、業務量が多いアプリケーションを担うシステム基盤については、未だにメインフレームが向いているとのことである。

   これは、メインフレームベンダーの意見だからというだけではないだろう。対象分野や業態変化の少ない業務に、数年単位でバージョンアップが必要となるオープン系技術を採用してしまった場合を想定してみる。業務上、バージョンアップすることには特段の利益もないのにもかかわらず、そ

---

[27] Return on investment の略であり、当該の投資がどれだけの利益を生んでいるのかを測る際に使われる基本的な指標であり、事業における投資の運用効率を示す。

[28] Business Intelligence の略であり、業務システムなどから蓄積される企業内の膨大なデータを、蓄積・分析・加工して、企業の意思決定に活用しようとする手法。

のための費用が定期的に必要となり、更に、一つのバージョンアップが周辺基盤ソフトウェア（例えばDB等）のバージョンアップを誘発することが多い。これは結果的に必要の無い出費が繰り返されるということになる。このような事態を想定する必要がある。

2. レガシーシステム移行はケースバイケースである。

同業他社での比較や横並び意識によってレガシーシステム移行のベストプラクティスが決まるのではなく、100の組織があれば100通り分のレガシーシステム移行の方法が存在する。なぜなら、レガシーシステム移行の目的はそれぞれ異なるはずであり、異なる目的に応じた移行方法が存在するからである。

業務や組織の特性を中心に検討すると、メインフレームを積極的に残すべき業務も存在し、また、オープン化したほうが効果的な変化の多い業務等も存在していることがわかる。これら全社的な業務の特性に鑑みてアーキテクチャ全体を捉えることが必要であるし、移行のコストを抑えるために、必要な部分はメインフレームを使い、そうで無い部分はオープンを使うという構成もありうる。移行コストが多く必要となっても、新たな価値を付加するためにすべてオープン技術を使うという選択もありうる。この様に目的に応じた移行を行うべきである。

3. レガシーシステム移行はステップバイステップである。

多くのレガシーシステムは数十年という期間を経て組み上げられたものである。従って、一気呵成に新しいシステムへ移行することは困難である。最初に移行方針を決めるための調査を実施し、既存プログラムの棚卸しをおこなう必要がある。ツールでの単純移行を選択したとしても、綿密な作業、そして打ち合わせと方針策定を必要とする。

従って、準備段階で十分な検討を行い、実現可能性や移行容易性等、コスト以外の要素にも考慮しながら長期的な視野で実施する必要があると言える。

4. レガシーシステム移行は短期的なコストテイクアウトではない。

最後のポイントは、レガシーシステム移行で短期的に劇的なコスト削減を実現することは困難なことである。

もちろん、ベンダーにロックインされていた状態に比べれば、競争原理が働く以上、調達のオープン化によって短期的なコスト削減を実現できる場合もある。しかし、たとえメインフレームであっても発注者側の管理によっては競争状態が発生するし、過度な費用請求が行われていると決め付けることはできない。もしベースのコストが適正であった場合は、単純にレガシーシステム移行を行うことによって直ちに大きなコストの削減は望めないとも言える。また、移行に必要な検討そのものや準備期間に発生するコストは決して少なくはないことも認識する必要がある。

NEC の事例に見られたが、オープン化で得られるコスト削減は長期的な観点で計画し、また判断することが重要である。そのために、富士通の事例では長期的な観点で ROI を計測することを必要としていた。長期的な視点での費用対効果を考える必要があるだろう。

以上、事例からわかった、レガシーシステム移行の実像についてまとめた。

ここにあるレガシーシステム移行事例における発見、もしくは共通点は、一般的に言われているイメージと異なっているかもしれないが、これが多くのレガシーシステム移行を手掛けているベンダーの経験から見えてくる事実でもある。

3.2 目的設定と達成評価

2章では、主に基幹系における高信頼技術やメインフレームの受け皿となる技術について、現状に関する調査・分析を行い、メインフレームとオープンの双方において高機能・高信頼性技術の多様性を確認し、移行における方法や多様化した技術の選択方法についてまとめた。

また、実際の移行事例を調査することで、レガシーシステム移行の実像に迫り、レガシーシステム移行が必ずしもオープン化技術の採用とは限らないことや、移行に際して期待するメリットに関して、綿密な検討を行いながら、理想的なアーキテクチャを段階的に形作っていく必要性を提示し、システム刷新の方向性についてまとめた。

これにより、情報システムの刷新には移行するかしないかも含めた移行前の検討ステップの充分な検討が、その成否を左右する重要な要素であると考える。また、移行判断後に、果たして判断が正しかったのかを検証することも同様に重要である。そこで、どのようなステップで移行を評価するべきか、という観点で、コスト削減の長期的把握の視点も含めて、移行後の評価方法について検討する。

3.2.1 移行判断前と移行判断後の評価の関係

これまで述べてきた通り、レガシーシステム移行における目的確認及びその設定は、システム移行における第一歩として捉え、この計画を綿密に行うことが望ましい。その意味で、移行の目的が達成されたときの想定効果とその効果が実際に得られたかについて検討するフレームワークについて事前に検討しておくことも必要である。

図 17　移行判断前と移行判断後における判断/評価のフレームワーク例

図 17 はそのフレームワークの一例とその現状に対する考察である。この図では、ユーザーの組織を民間と官公庁に分けたうえで、移行判断に際してどのような検討を行っているのか、評価しているのかを整理したものである。

事例でも確認することができたが、民間のレガシーシステム移行判断前においては、移行方法を検討したり、プログラムの棚卸しが行われているが、官公庁ではそれに代替する調査として、移行することを前提とした刷新可能性調査[29]が行われている。一方、移行判断後に関しては、民間では移行後に得られる絶対性能の評価、負荷等の評価が行われるのに対して、官公庁ではシステム移行の評価そのものというよりは、調達のみの評価としてコスト削減と分割方式の有無等が重視されており、民間での評価軸とは大きく異なる。

[29] 最適化計画の策定のための予備調査として位置付けられる、主にレガシーシステムの刷新の現実性について、コスト、移行期間や方式等について検討し、利便性を損なうことなく想定するコスト（導入コスト、運用コスト×想定利用年数）を下げることができるか否かについて検討し，結論を得ることが目的とされている。

これまで、移行判断前にどのような評価が民間で行われているかという点については十分触れてきたので、以下では移行判断後において民間でどのような評価が行われているかについて記述する。

3.2.2 移行後における評価と評価指標

そもそも、移行判断後と移行後の評価というのはどのようなタイミングで行えばよいのだろうか。以下の図 18 では、システム移行における業者決定から本番稼動までの時間軸を表している。

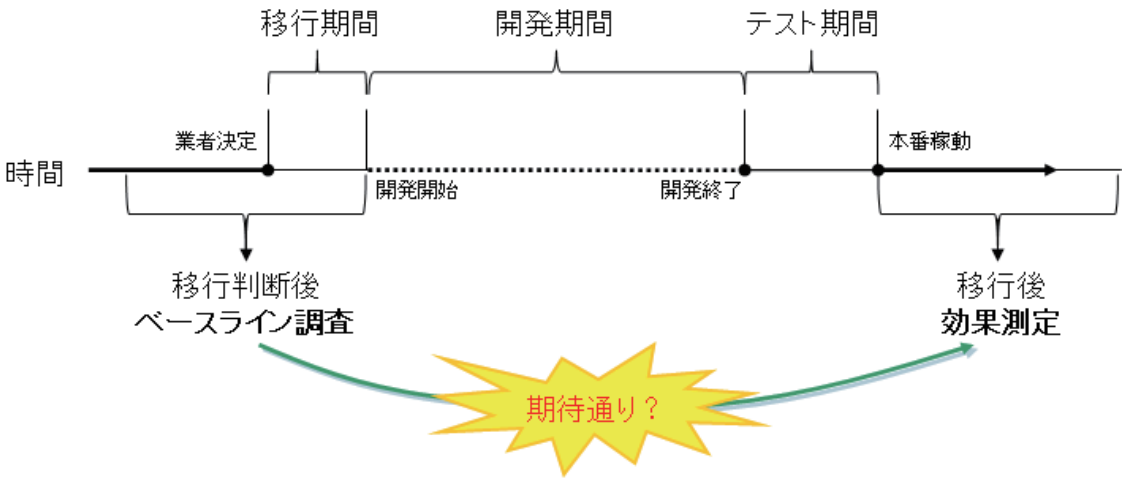


図 18　移行判断後と移行後の評価タイミング

この時間軸の中で、まず開発を開始するにあたって、現状のシステムの状況を棚卸しすることになる。その際にベースライン調査として、評価を行うための起点を定めるための現状調査を行う。この現状調査の元となる指標については、理想的には、常時収集している数値（後述する技術面、目的面、投資面等）の平均値をベースラインと設定することが望ましい。仮に常時収集していなかったら、調達開始から実際に開発が開始するまでのタイミングでも収集しておくことが必要となる。

次に、システムの移行（開発）が終了した後、本番稼動まではテスト期間であることが多い、このテスト期間が終了するまではシステムの評価を行わず、実際に本番稼動した際に、改めて移行後の評価を開始することになる。ただ、移行後の評価を本番稼動開始後のどれくらいで行うかは議論の余地があるところではある。通常、バグやシステムの小規模改修が落ち着いてくるのは規模にもよるが半年から１年程度であるので、更に 1 年程経過したところで評価を行うことが現実的かもしれない。

その際には、小規模改修を含んだケース（約 2 年）と小規模改修が終了したケース（約 1 年）等という形で、評価数値を算出して比較する方法もありうる。

次に、移行判断後と移行後における評価については、どのような指標を用いて評価するべきであろうか。一般的には、システム移行（導入）を評価するに当たり、目的への合致性を評価することはもちろんであるが、図 19 の様に、導入した技術が適切なパフォーマンスを発揮しているかという技術面及び適切な技術が適切なコストで導入されているかという評価と、導入によって運用コストがどのように変化したかについても評価し、多様な視点からの総合評価を行うことが必要である。

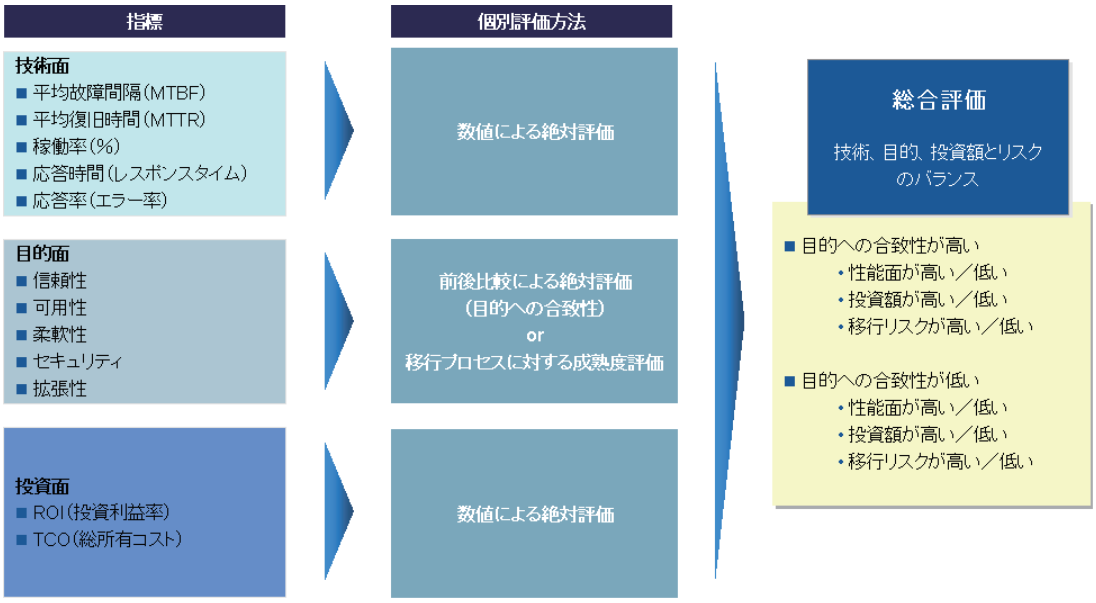


図 19　評価指標の例について

主に技術面では、MTBF[30]や MTTR、[31]そして稼働率等を中心としたシステムの性能について、図 19 のように移行判断後と移行後で定量的に評価を行うことにより、移行後に著しい性能低下がないかどうかを判断することができる。著しい性能低下

[30] Mean Time Between Failure の略であり、日本語では平均故障間隔と訳される。単独の機器や複合的なシステムが故障するまでの時間の平均値であり、使用を開始して、故障が起こり、その故障から回復してから、次に故障するまでの平均時間をいう。

[31] Mean Time To Repair の略であり、日本語では平均復旧時間と訳される。故障したシステムの復旧にかかる時間の平均であり、「修理時間の和÷故障回数」として計算される。

があれば、移行作業自体は成功であっても、移行におけるプラットフォームや製品において、十分な性能を発揮できる選択をしなかった可能性が高い。

一方、目的面では主に移行後の運用における柔軟性や拡張性が移行判断後の狙い通りに発揮されているかについて主に評価する。また、業務的な課題があり、リエンジニアリング等の手法で導入することにより、その課題の解消が目的で移行が行われたのであれば、その目的が達成されたのかについても判断する必要がある。この目的面の評価は定性的になりがちである。

また、投資面での評価では、ROI や TCO[32]を評価する必要はある。必ずしも投資に対して金額的リターンを有するシステムだとは限らないので、TCO だけを評価するケースももちろんあって良い。ROI を評価することによって、投資とリターンのバランスを確認できる。例えば、投資が大きいがリターンが少ない場合は、投資が目的と合致していたのか、目的面とあわせて原因を検討することになる。TCO に関しては、図 20 のように移行判断後と移行後のコスト状況を明確にすることができ、移行後に思わぬコスト増加が見られる等の状況を把握することができる。また、TCO は単なるハードウェアやソフトウェアだけでなく、付随する施設や要因の人件費も含めた、総体的なコストを把握する概念であるので、移行判断後と移行後の把握によって、実際のコストがどのように変化したか、例えばハードウェアは安くなったが要員人件費が増えたといったようなコスト変動が把握できるのである。

特にTCOの考え方を用いることで、1章で触れたようにメインフレーム環境からオープンシステムに移行した時に代表されるように、単純にシステムに関わる開発や運用の費用だけでなく、いわゆる「見えないコスト」になりがちな、複数のベンダーの契約や調整に関わるコストについても把握したうえで、総額の変化を把握することが出来る。

[32] Total Cost of Ownership の略で、総所有コストと訳される。ある資産や設備等に関する、購入から廃棄までに必要な時間と支出の総計で算定される。単に資産としての購入コストだけではなく、保守・運用等に関わるコストも含んだ形で算定する。

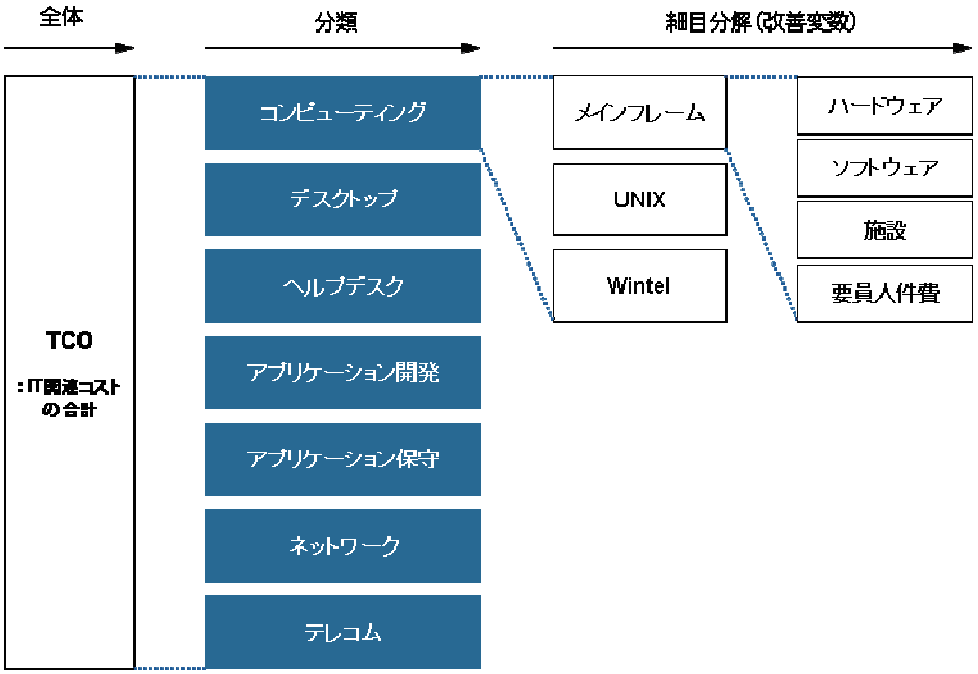


図 20　TCO の概念（出典：ガートナー社）

最後に、総合評価を行うことも重要である。例えば、性能面と目的面の評価が高くても、TCO が著しく上がっているということになれば、改めてコスト増と性能、目的両面の向上のバランスが適切なものなのか、検討する必要がある。場合によっては、性能面を落とすことで、TCO も下げるという調整を可能な状態としておかなければならない。逆に、TCO は低くなったが、性能、目的両面の評価が低いものになれば、コスト削減だけが目的だったのか、改めて検討する必要がある。これも場合によっては、TCO が上がっても、性能面の向上が必要であることもあるだろう。

以上、民間企業での移行後の評価方法について紹介した。移行判断後と移行後の評価を行うことにより、レガシーシステム移行そのものを目的とするのではなく、移行による効果を想定しながら、実際に得られた効果を計測することによって、本来のレガシーシステム移行の効果をより享受できるはずである。

3.3　システム調達における実際

3.3.1　システム調達における評価方法についての課題

ここでは、これまで紹介した民間企業での評価方法と対比して、官公庁で行われている移行判断後に行われる評価について、官公庁のシステム調達を中心にベンダー企業から収集した官公庁における調達の実際についても踏まえ、その評価方法の課題について取りまとめる。

官公庁におけるシステム調達は、図 21 のように大きく 3 つのステップに分かれる。

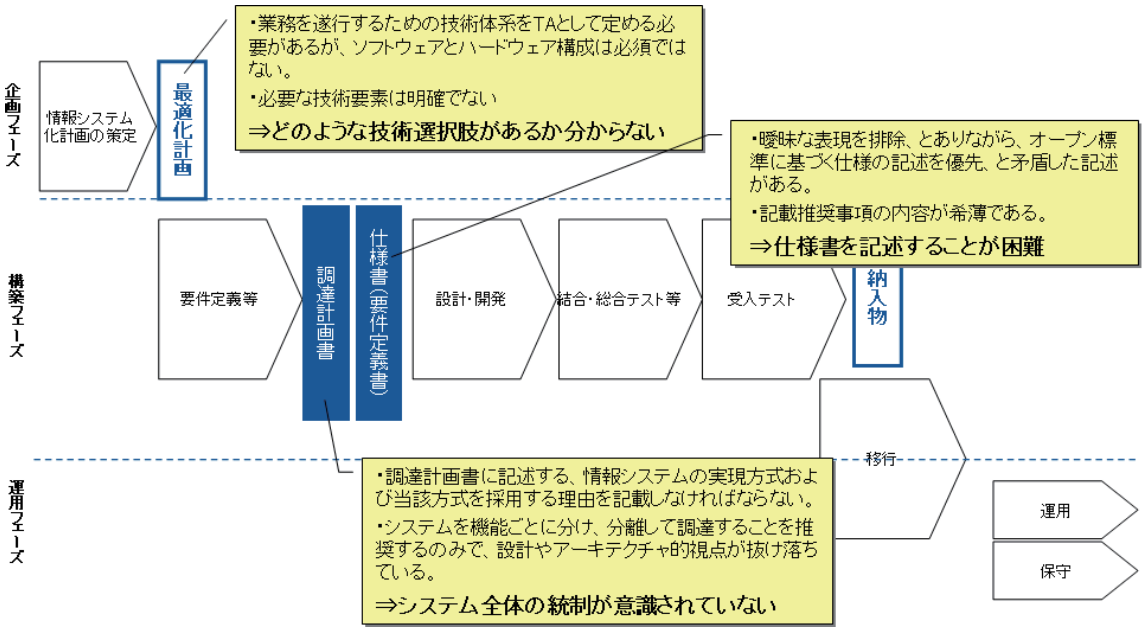


図 21　調達プロセスにおける問題点

企画フェーズにおいては、主に情報システム化計画の策定を行い、その流れで最適化計画を行い、アーキテクチャ視点から省庁間を含めて、システム機能の重複を出来るだけ省き、洗練されたシステムを構築することを目的としている。

ベンダー企業から多く聞かれた意見では、エンタープライズアーキテクチャの手法を用いた業務フローの記述とその業務の最適化の取り組みについては、業務の可視化等の一定の効果が出ているが、具体的にそれら業務をどのような範囲で括

り、どのようなアプリケーション、そして技術要素で実現するのかに関するつながりが無く、結果的に技術要素を選択することが難しいという指摘であった。実際には、TA（Technology Architecture）として技術リファレンス等を充実させる取り組みが進展しているのだが、業務との関わりや活用方法が理解できていないため、結果として何を選択したら良いのかが不明確になっているようである。

この様な状態であれば、これまで論じてきたようなレガシーシステム移行のプラットフォームを検討し、選択することは難しくなってしまうと言えよう。従って、初めからオープン技術の採用等と方針を明示したほうが調達が容易になるとも考えられよう。

次に、構築フェーズにおいて大まかに要件定義を行った後、調達計画書を策定することになるのだが、この調達計画書においては、分離して調達することに目的が集中してしまうことによって、システムにおいて重要なアーキテクチャ、全体としてどのように機能を実現していくかという視点に欠けてしまい、結果として、開発の現場でシステム間の調整作業に手間がかかっているという意見が多く聞かれた。具体的にこの問題に対処するためには、官公庁でもアーキテクトに近い、組織のシステムアーキテクチャの維持と管理を担う人材が必要となるであると考える。また、民間企業においてはベンダーにロックインされることを容認しながらも、アーキテクチャの維持と管理についてベンダーにある程度任せてしまうという手法もあるとされるが、この方法を直ちに官公庁に当てはめることは公平性の観点から困難である。この様に、分離調達を過度に推進するが故に、アーキテクチャ維持の問題も生じてきているが、その対処方法についても解が得られておらず、そもそもの分離調達の目的と実態の乖離について再考察する必要があるだろう。

特に、2章で論じたように、システムは個別のシステムが別々に動作するというよりは、個々のシステムが連携し合い、全体として機能を発揮する方向に進みつつあるため、この様な調達手法では市場の技術動向を反映できず、選択できる方法に制限が出てしまうことが懸念される。

また、同じ構築フェーズでの後続のステップである調達仕様書における課題について、仕様が曖昧であり、考え方によってはいかようにでも解釈できる仕様書の記述が多いという指摘が多かった。その原因として、政府における仕様書作成のガイドラインには、曖昧な表現を排除するとありながら、オープン標準に基づく仕様の記述を優先し特定の技術等を記載しないといった矛盾した記述があるだけでなく、

記載推奨事項の内容が希薄であるといったような、現在のガイドラインそのものが未成熟であるとの根本的な指摘もあった。

仕様書が不明確であれば、ベンダーが提案するシステムの移行方法やシステムに関しても曖昧なものになりがちである。設計上はオーバースペックにしておいて性能や信頼性の余裕を多くとることで、リスクを回避しようとするベンダーの動きにつながるだろう。結果として、曖昧な仕様でシステムを構築することになり、高コストになる可能性があるのではないだろうか。

4. まとめ

本調査研究においては、レガシーシステム刷新に代表される情報システムの刷新について、主な論点として技術面とガバナンス面からの調査分析を実施した。特に最新の技術動向についてはガートナー社のリサーチ情報を活用すると同時に、当研究所内で「行政における基幹系システム・フレームワークに関する研究会」を実施することで、最新情報に基づいた分析となるように留意した。またガバナンス面においては事例に基づいた成功要因を抽出するとともに、その暗黙知を形式知に転換し汎用的なツールとして活用するというアプローチを試みた。

その結果について以下にまとめる。

4.1 技術面から見た情報システム刷新のあり方

2 章及び 3 章では、まずエンタープライズサーバーやその周辺製品の技術動向を確認し、レガシーシステム刷新後のシステム像について調査研究を実施した。

政府情報システムにおけるレガシーシステム刷新に着手した数年前と比較し、様々なアーキテクチャを活用することで移行対象として利用できる、いわゆる「受け皿」となる技術や製品が多方面において大きく進化していることがわかった。特に信頼性については、サーバー単体での進化だけでなく、周辺技術との組み合わせによる情報システム全体としての高信頼化という方向性が明確となっている。これは 3 章に記述した研究会においてITベンダーより提示された事例にも表れている。また同時にメインフレームにおいてもオープン化対応の流れがあり、市場に再評価されていることも判明した。

アーキテクチャを選定するにあたっては、業務特性から導き出せる機能要件だけではなく、情報システムにおいて運用面や性能面で実現するべき機能である、非機能要件についても明確化が必要である。その明確化の手法は一つではないが、本調査研究においてはユーザー部門の立場からアーキテクチャ選定を可能とする手法を提案した。

この手法によって、アーキテクチャだけでなく、具体的なソリューションに落とし込んだうえで、情報システムプラットフォームのあり方や実現するべき機能についても踏み込むことができた。

4.2 ガバナンス面から見た情報システム刷新のあり方

3 章では事例を中心に、情報システム刷新におけるガバナンスの重要性とそのポイントについて調査分析を実施した。

多くの事例において共通的に見られたのは、移行目的と移行判断の重要性である。そもそも移行するべきかどうか、移行する際にはどのような手法を活用するべきかという判断において、ユーザー自らがシステム面だけでなくビジネス面も考慮し、情報システムの価値の検討を経て決定している点である。これは言い換えれば、移行判断や移行手法は多種多様であって、業界や他社の動向に流されるのではなく、各々が当事者として判断し選択することの重要性を示しているとも言える。

また移行の成否を判断する際には、移行作業そのものはあくまでも過程であり、移行目的に照らして移行後の価値を測定することが重要であることにも言及した。あえて初期コストを投じて移行に踏み切る以上、移行後にはその恩恵を得られなければならない。特に運用管理においては、実際に稼動してからでなければ把握できなかった労力やコストが発生する可能性もある。従って所期の目的を達成できたかどうかを例えば TCO 等の指標を用いて総合的に評価する必要があるだろう。具体的には TCO を用いた評価が挙げられる。このような視点での効果測定のあり方について検討を実施した。

そのうえで、情報システムの調達について現状把握と分析を行い、上記ガバナンスに適応した調達をどのように行うべきか検討した。

4.3 政府情報システムにおけるシステム刷新のあり方

技術面から得られた事実からは、レガシーシステムの刷新にあたっては、オープン化ありきではなく、業務やシステム上の処理特性を把握し、それぞれに適したアーキテクチャを採用するべきとの結論を得た。更に、各ベンダーのインタビュー結

果を考察することによって、ユーザー部門の立場からアーキテクチャ選定を可能とする手法を提案した。これによって製品ありきではないアーキテクチャ選定が可能となるだろう。この手法は、より具体的な情報システムのグランドデザインにおいても役に立つと思われる。

ガバナンス面から見た考察では、実際の移行事例を調査し、移行において効果的な施策やポイントを提示した。いくつかの課題が明らかとなったが、結論として特に重要なのは、移行判断に向けた最新の情報を収集する活動と、仕様書における要件の詳細化と明確化であった。

これら前述の考察に加え、政府情報システムの調達プロセスを意識し、課題の抽出及び対策の検討を実施した。

まず、IT 市場や民間企業の動向を常に確認し、採用するべき技術やガバナンスについて情報収集を行い、自らの計画を定期的に見直すことが必要である。その際には、新しい技術をやみくもに追いかけるのではなく、自らの業務やシステムにとって有益な技術を見出し、事例等から実現可能性について十分な検討を行うことも必要である。

さらに、政府においては分離分割調達という難易度の高い調達手法を前提としているため、人材育成を含めた経験や知識の内部留保が重要な課題であることも再認識した。人材面から見ると業務とシステムの両面を理解していることが理想的ではあるが、分離調達を前提としているため、アーキテクチャについても理解しグランドデザインを描くことができる人材が必要である。今回行った調査の過程においても、場当たり的なオープン化を危惧する声があった。グランドデザインを描くことができる人材の確保が、発注者側において重要視されるべきであろう。

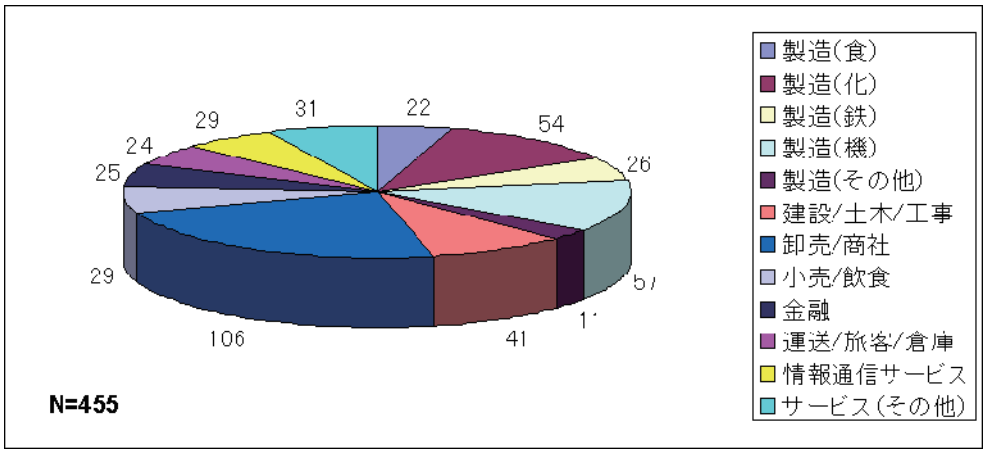
将来的には、レガシーシステムの刷新がどのような効果を生み出したのかについても測定する必要がある。最適化計画の効果指標においては、システム経費の削減が重要視されているが、この経費の範囲についてより広く捉える必要があると言えよう。具体的には、職員側の発注や運用に係る稼動も算入する必要があるはずである。さらに、信頼性や性能等についても、新たな技術の恩恵を具体的な効果として得られたのか、本来であれば検証する必要があると思われる。

政府情報システムの刷新を進めていくにあたっての課題点を通じて、最適な方法を探る調査研究を実施してきた。これまで指摘したように技術の進展に伴う多様な選択肢に目を配り、業務やシステムにもっとも適合するような刷新方法を取ることが肝要である。それら留意点を踏まえたうえで、単なるオープン化に傾きがちな従来の思考から脱却し、多様な選択肢の分析を行い、刷新を実行した後も予め設定した評価指標に基づき適宜評価を加える。そして、さらに適切な手法を導入するといったサイクルを確立することが必須である。その継続が、最適な政府情報システムを実現させると言えよう。

5. Appendix

5.1 図 6 （複数選択）メインフレームのオープン系システムへの移行状況

本図で用いられたグラフにおける調査対象となった企業の内訳を以下に示す。



出典：2008 年 11 月ガートナー社調査より

5.2 参考文献一覧

- ガートナーリサーチ　JEPS-03-28　Sun のリホスティングの事例に学ぶメインフレーム移行のベスト・プラクティス（April 25,2003）
- ガートナーリサーチ　G00147519　アプリケーション・インフラストラクチャのマジック・クアドラント：2007 年第 2 四半期（January 31, 2008）
- ガートナーリサーチ　G00136909　主要 IT ベンダーの Linux 対応とオープンソース・コミュニティへの貢献（June 5, 2006）
- ガートナーリサーチ　進まないレガシー・マイグレーション：その将来はクリティカルである（October 10, 2007）
- ガートナーリサーチ　日本におけるミッドレンジ／ハイエンド NAS ソリューションのマジック・クアドラント：2008 年（January 30, 2009）
- ガートナーリサーチ　日本における企業向けハイエンド／ミッドレンジ Unix サーバ、Itanium サーバ製品のマジック・クアドラント：2007 年（January 25, 2008）
- ガートナーリサーチ　日本のサーバ・ベンダーはハイエンド・サーバ戦略を再考すべきである（December 10, 2008）

- ガートナーリサーチ　G00165017 A Summary of Magic Quadrants for Application Infrastructure（3 March,2009）
- ガートナーリサーチ　G00153357　Magic Quadrant for Worldwide Server Vendors（21 December ,2007）
- ガートナーエクゼクティブプログラムリポート　ハイバリュー･ハイリスクにフォーカスしてレガシー・ポートフォリオを管理する（September, 2006）
- 日経 BP 社　レガシーマイグレーションへの挑戦―Mainframe to open（September,2003）
- リックテレコム　メインフレーム実践ハンドブック z/OS（MVS）,MSP,VOS3 のしくみと使い方　神居 俊哉/著, 高尾 司/監修（March,2009）
- 翔泳社　アーキテクトの審美眼　萩原正義/著（March,2009）
- ワーキングペーパー　戦後日本の行政能率政策と電子計算機の登場　奥村裕一著（2007）
- 行政における電子計算機導入の実態と問題点―Ⅰ―　社団法人　行政事務機械化研究協会著　行政と ADP 創刊号(1963)

5.3　参考資料:基盤選定テンプレート

区分	要件No.	回答必須 or 任意	内容	選択肢	備考
1.システム概要	1.1	必須	システム特性	■基幹系システム □社外向けWebシステム（参照系） □社外向けWebシステム（更新系） □社内向けWebシステム（参照系） □社内向けWebシステム（更新系） □C/Sシステム	
	1.2	必須	DB使用形態	■更新系（OLTP） □参照系(OLAP）	
	1.3	任意	システム再利用	□既存資産の転用、相乗りの可能性有 ■既存資産の転用、相乗りの可能性無	
	1.4	任意	システム導入実績	□類似システム導入実績経験有 ■実績なし	
2.システム稼働	2.1	必須	サービス稼働時間	■365日24時間（計画停止不可） □365日24時間（計画停止可） □8:00～24:00 □業務時間内	
	2.2	任意	オンライン系処理サービス稼働時間	■365日24時間（計画停止不可） □365日24時間（計画停止可） □8:00～24:00 □業務時間内	
	2.3	任意	バッチ系処理実施許容時間	□バッチ処理無し ■実施許容時間：3時間	
	2.4	任意	バッチ系処理実施時間帯	■実施許容時間帯：0：00～3：00	
	2.5	必須	計画停止可能時間	■停止不可 □30分以内 □1時間以内 □3時間以内 □6時間以内	サービス稼働時間が計画停止不可の場合、自動的に停止不可
	2.6	任意	データバックアップに伴うサービス停止許容時間	■停止不可（オンラインバックアップ） □停止可能時間：＿＿＿＿＿＿＿＿	
	2.7	任意	メンテナンスに伴うサービス停止許容時間	■停止不可 □停止可能時間：＿＿＿＿＿＿＿＿	
	2.8	任意	法定点検に伴う計画停止の有無	□有　（頻度：＿＿＿＿＿＿＿＿） ■無	
	2.9	任意	計画停止可能時間帯	□停止可能時間帯：＿＿＿＿＿＿＿＿	
	2.10	任意	データバックアップに伴うサービス停止可能時間帯	□停止不可（オンラインバックアップ） □停止可能時間帯：＿＿＿＿＿＿＿＿	
	2.11	任意	メンテナンスに伴うサービス停止可能時間帯	□停止不可 □停止可能時間帯：＿＿＿＿＿＿＿＿	
	2.12	任意	法的点検に伴う計画停止の有無	□有 □無	
	2.13	必須	システム障害からの復旧許容時間（MTTR）		予備機等への切替時間など
	2.14	任意	H/W障害からの復旧許容時間		
	2.15	任意	DB障害からの復旧許容時間		
	2.16	任意	N/W障害からの復旧許容時間		
3.運用サービス要件	3.1	任意	統合運用・監視サービス	□統合運用・監視サービスを利用 □利用しない	
	3.2	任意	統合バックアップサービス	□統合バックアップサービスを利用 □利用しない	
	3.3	任意	マネージドストレージサービス	□マネージドストレージサービスを利用 □利用しない	
4.利用ユーザー	4.1	必須	利用対象ユーザー	□組織外不特定ユーザー □組織外特定ユーザー □組織外不特定ユーザー □組織外特定ユーザー	
	4.2	必須	同時利用ユーザー数		
	4.3	必須	最大利用ユーザー数		
	4.4	任意	同時利用ユーザー数（ピーク）		
	4.5	任意	同時利用ユーザー数（平時）		

区分	要件No.	回答必須 or 任意	内容	選択肢	備考
5.バックアップ	5.1	任意	バックアップタイミング（Webサーバー）	□オンライン・バックアップ □オフライン・バックアップ □バックアップ不要	計画停止不可の場合、バックアップタイミングはオンライン・バックアップ選定
	5.2	任意	バックアップタイミング（APサーバー）	□オンライン・バックアップ □オフライン・バックアップ □バックアップ不要	計画停止不可の場合、バックアップタイミングはオンライン・バックアップ選定
	5.3	任意	バックアップタイミング（DBサーバー）	□オンライン・バックアップ □オフライン・バックアップ □バックアップ不要	計画停止不可の場合、バックアップタイミングはオンライン・バックアップ選定
	5.4	任意	バックアップ時のパフォーマンス劣化	□ツールによるパフォーマンス劣化回避 □パフォーマンス劣化不可	オンラインバックアップを利用する場合
	5.5	任意	バックアップ・システム構成	□ミラーディスク切り離し型 □スナップショット型 □NAS型 □ネットワーク経由型 □テープ装置直結型	
	5.6	任意	バックアップ対象データ	□全領域 □システム領域（OS）＋データ領域 □データ領域のみ □システム領域（OS）のみ □バックアップ不要	
	5.7	任意	バックアップデータ保管形態	□ディスク装置 □テープ	
	5.8	任意	バックアップ方法	□フルバックアップ （頻度：＿＿＿＿＿＿＿＿） □増分バックアップ （頻度：＿＿＿＿＿＿＿＿） □差分バックアップ （頻度：＿＿＿＿＿＿＿＿） □バックアップ不要	統合バックアップサービスを利用する場合、管理者が最適な方法を選択
	5.9	任意	テープの外部保管	□テープの外部保管利用 □テープの外部保管利用なし	
	5.10	任意	バックアップデータ保管世代数	□＿＿＿＿＿世代	
	5.11	必須	障害時のリストア処理に許容される時間（RTO）		
	5.12	任意	DBデータのリストア処理許容時間		
	5.13	任意	システムデータ（OS、S/W）のリストア処理許容時間		
	5.14	必須	障害時のデータ復旧ポイント（RPO）		
	5.15	任意	DBデータの復旧ポイント		
	5.16	任意	システムデータ（OS、S/W）の復旧ポイント		
6.システム運用	6.1	必須	サーバー設置場所	□データセンター □ビル内サーバールーム □ビル内フロアー	
	6.2	任意	運用担当者の保有スキルとの合致	□未導入製品　なし □未導入製品　あり	
	6.3	任意	リモート管理の必要性	□アプリケーションが必要 □ターミナルサービス（Windows） □telnet □リモート管理不要	
	6.4	任意	開発環境構成	□開発+検証＋本番 □開発＆検証＋本番 □開発＆検証＆本番	
	6.5	任意	開発に携わるユーザ数		
	6.6	任意	開発環境の利用期間		
	6.7	任意	開発環境データ、プログラムのバックアップの必要性	□あり（プログラム、マスタ、トランザクション） □あり（プログラム、マスタ） □あり（プログラムのみ） □なし	
7.システム監視	7.1	必須	稼働監視の監視レベル	□死活監視 □リソース監視 □プロセス監視 □ジョブ監視 □ログ監視 □データベース監視 □（ポート監視） □（Web接続監視） □監視対象外	運用サービス要件と連動
	7.2	必須	監視体制	□365日24時間　有人監視 □業務時間内　有人監視+夜間オンサイト □オンサイト □担当者による監視	運用サービス要件と連動
	7.3	任意	アラート通知方法	□電話による担当者通知 □メールによる担当者通知 □なし	運用サービス要件と連動

区分	要件No.	回答必須 or 任意	内容	選択肢	備考
8.サポート	8.1	任意	ハードウェアベンダー保守契約の内容	□24時間365日対応 □平日9:00～18:00のみ □なし	
	8.2	任意	ハードウェアベンダーのサポート体制	□常駐（人・保守品） □オンサイト（駆け付け×時間以内） □センドバック □なし	
	8.3	任意	ソフトウェアベンダー保守契約の内容	□24時間365日対応 □平日9:00～18:00のみ	
	8.4	任意	ソフトウェアベンダーのサポート体制	□常駐（人） □オンサイト（駆け付け×時間以内） □センドバック □なし	
9.システム冗長化	9.1	任意	H/Wクラスタ構成の必要性 （Webサーバ）	□Active-Standby（完全Standby型） □Active-Standby（相互Standby型） □負荷分散型 □不要	
	9.2	任意	H/Wクラスタ構成の必要性 （APサーバ）	□Active-Standby（完全Standby型） □Active-Standby（相互Standby型） □負荷分散型 □不要	
	9.3	任意	H/Wクラスタ構成の必要性 （DBサーバ）	□Active-Standby（完全Standby型） □Active-Standby（相互Standby型） □負荷分散型 □不要	
	9.4	任意	OS、S/Wクラスタ構成の必要性	□必要 □不要	
	9.5	任意	DBクラスタ構成/高可用性の必要性	□必要 □不要	
	9.6	任意	RAID（サーバローカル）	□ミラーリング（RAID-1） □分散＋パリティチェック（RAID-5）	
	9.7	任意	RAID（ストレージ）	□ミラーリング（RAID-1） □分散＋パリティチェック（RAID-5） □ストライプ・ミラー（RAID-10）	
10.システム性能	10.1	任意	負荷分散構成の必要性 （Webサーバ）	□LoadBalancer利用 □不要	
	10.2	任意	負荷分散構成の必要性 （APサーバ）	□必要 □不要	
	10.3	任意	負荷分散構成の必要性 （DBサーバ）	□Oracle RAC利用 □不要	
11.1データ連携	11.1	任意	他システムとのデータ連携の頻度と容量		
	11.2	任意	相手システムの選択可能なデータ連携方式	□FTP □NIS-MAIL □ファイル共有 □その他（　　）	
	11.3	任意	データ連携のリアルタイム要件の有無	□あり □なし	
	11.4	任意	データ連携におけるデータ加工の必要性とその方法	□あり（専用ツールを利用） □あり（専用プログラムを開発） □あり（DB標準機能で対応） □なし	
12.ミドルウェア	12.1	任意	ミドルウェアに要求される機能		
	12.2	任意	Serverのバージョン		
	12.3	任意	セッションタイムアウト		
	12.4	任意	クラスタ構成の必要性	□クラスタ不要 □HttpSessionオブジェクト □ステートフルセッションBean	
	12.5	任意	アプリケーションの要件		
	12.6	任意	Server以外の負荷発生要件 （同居するシステムの有無）		
	12.7	任意	インスタンスの構成		
	12.8	任意	クラスタの構成の必要性と構成方法		
	12.9	任意	負荷分散機能の必要性と構成方法	□ハードウェアロードバランサ □Webサーバプラグイン	
	12.10	任意	WebコンテナとEJBコンテナの分離の必要性と分離方法		
	12.11	任意	管理サーバの構成方法	□管理サーバ/1サーバ □管理サーバ＋管理対象サーバ/1サーバ □管理サーバ兼管理対象サーバ/1サーバ	
	12.12	任意	帳票要件（頻度、容量、出力型式）		
13.ネットワーク	13.1	任意	WAN冗長化の必要性	□必要	
	13.2	任意	LAN冗長化の必要性（データセンター）	□必要	
	13.3	任意	LAN冗長化の必要性（バックアップ）	□バックアップ専用LAN必要 □不要	
	13.4	任意	NIC冗長化の必要性とその方式	□必要（Load Balance） □必要（Active-Standby） □不要	

区分	要件No.	回答必須 or 任意	内容	選択肢	備考
14.セキュリティ	14.1	任意	ログ管理の活用方法		
	14.2	任意	サーバへのログオン/ログイン制御方式	□アプリケーション共通基盤ユーザーテーブル使用 □制御不要	
	14.3	任意	ユーザ情報（アカウント、権限）の格納方式	□データベースサーバ	
	14.4	任意	サーバリソース（ファイル、アプリケーション）へのアクセス制御の必要性と実現方法	□ACL（Access Control List）設定 □アプリケーション認証 □制限不要	
	14.5	任意	アクセスログ取得の必要性と実現方法、保管方法	□専用ツール導入 □OS標準監査設定+カスタムメイド機能追加 □OS標準監査設定 □アクセスログ不要	
	14.6	任意	パケットフィルタリングの必要性と実現方法	□ネットワークアプライアンスで実装	
	14.7	任意	データ暗号化の必要性と実現方法	□必要	
	14.8	任意	データ持ち出し対策の必要性と実現方法	□必要	
	14.9	任意	ウィルス対策の実現方法	□必要	
	14.10	任意	外部アクセスの通信暗号化	□必要	
サイジング参考指標	-	必須	参照処理のレスポンスタイム		
		必須	更新処理のレスポンスタイム		
		任意	DB参照処理レスポンスタイム		
		任意	DB更新処理レスポンスタイム		
		必須	1日のトランザクション量		
		任意	ピークのトランザクション量		
		任意	1日のDBアクセス数（クエリ数）		
		必須	利用ユーザー数の年増加率		
		必須	トランザクション量の年増加率		
		必須	DBデータ量の年増加率		
		任意	オンライン時の想定データ転送容量		
		任意	バッチ処理の想定データ転送容量		
		任意	システム利用期間内で想定される最大搭載CPU数		
		任意	システム利用期間内で想定される最大搭載メモリ容量		
		任意	バーストトラフィックの発生の有無、容量と対策	□月次処理時（第×週第？営業日） □?時～？時において？MB程度 □バースト性なし	
		任意	サーバ拡張性の確保方針（Webサーバ）	□スケールアウトでの実現 □スケールアップでの実現	
		任意	サーバ拡張性の確保方針（APサーバ）	□スケールアウトでの実現 □スケールアップでの実現	
		任意	サーバ拡張性の確保方針（DBサーバ）	□スケールアウトでの実現 □スケールアップでの実現	
		任意	OS格納に必要なサイズ		
		任意	アプリケーションモジュール、実行ファイル格納に必要なサイズ		
		任意	データベースデータ格納に必要なサイズ		
		任意	データベースログ格納に必要なサイズ		
		任意	バックアップデータ格納に必要なサイズ		

システム構成要素	要件No.	必須or任意	区分	内容	選択肢	対象	Essential	High	Middle	Low	備考
システム全般	1.1	必須	システム概要	システム特性	■基幹系システム □組織外向けWebシステム（参照系） □組織外向けWebシステム（更新系） □組織内向けWebシステム（参照系） □組織向けWebシステム（更新系） □C/Sシステム		■基幹系システム →可用性重視 □組織外向けWebシステム（参照系） □組織外向けWebシステム（更新系） →セキュリティ重視	□組織外向けWebシステム（参照系） →セキュリティ重視 □組織内向けWebシステム（更新系） →性能重視	□組織内向けWebシステム（参照系） □組織内向けWebシステム（更新系） □C/Sシステム →性能重視	□組織内向けWebシステム（参照系） □C/Sシステム →性能重視	
	4.1	必須	利用ユーザー	利用対象ユーザー	■組織外不特定ユーザー □組織外特定ユーザー □組織内不特定ユーザー □組織内特定ユーザー	OS、プラットフォーム	■組織外不特定ユーザー	□組織外特定ユーザー	□組織内不特定ユーザ	□組織内特定ユーザ	
	2.1	必須	システム稼働	サービス稼働時間	■365日24時間（計画停止不可） □365日24時間（計画停止可） □8:00～24:00 □業務時間内	バックアップ	■365日24時間（計画停止不可）	□365日24時間（計画停止可）	□8:00～24:00	□業務時間内	
	2.13	必須		障害からの復旧許容時間（MTTR）	1時間以内	バックアップ	5分以内	30分以内	1時間以内	1時間以上	
	－	必須	サイジング参考指標	参照処理のレスポンスタイム	5秒以内	負荷分散	3秒以内	5秒以内	8秒以内	9秒以上	参考：「米国のZona Researchの調査によると、Webのレスポンスタイムが7秒から8秒になるだけで、10%から30%のユーザを失い、12秒になると70%のユーザを失うというレポートがされています。米国ではこれを8秒ルールと称しているようです。」
	－	必須		更新処理のレスポンスタイム	5秒以内	負荷分散	3秒以内	5秒以内	8秒以内	10秒以上	8秒をMiddleレベルで設定
APサーバー	2.1	必須	システム稼働	サービス稼働時間	■365日24時間（計画停止不可） □365日24時間（計画停止可） □8:00～24:00 □業務時間内	OS、プラットフォーム バックアップ クラスタ構成	■365日24時間（計画停止不可）	□365日24時間（計画停止可）	□8:00～24:00	□業務時間内	
	2.13	必須		障害からの復旧許容時間（MTTR）	1時間以内	バックアップ	5分以内	30分以内	1時間以内	1時間以上	
	－	必須	サイジング参考指標	参照処理のレスポンスタイム	5秒以内	負荷分散	3秒以内	5秒以内	8秒以内	9秒以上	
	－	必須		更新処理のレスポンスタイム	5秒以内	負荷分散	3秒以内	5秒以内	8秒以内	10秒以上	
DBサーバー	1.2	必須	システム概要	DB使用形態	■更新系（OLTP） □参照系(DWH、OLAP、データマイニング）	DBMS	■更新系（OLTP）	■更新系（OLTP） □参照系(DWH、OLAP、データマイニング）	■更新系（OLTP） □参照系(DWH、OLAP、データマイニング）	□参照系(DWH、OLAP、データマイニング）	
	4.1	必須	利用ユーザー	利用対象ユーザー	□組織外不特定ユーザー □組織外特定ユーザー □組織内不特定ユーザー □組織内特定ユーザー	OS、プラットフォーム	□社外不特定ユーザー	□社外特定ユーザー	□社内不特定ユーザ	□社内特定ユーザ	
	2.1	必須	システム稼働	サービス稼働時間	□365日24時間（計画停止不可） □365日24時間（計画停止可） □8:00～24:00 □業務時間内	バックアップ	□365日24時間（計画停止不可）	□365日24時間（計画停止可）	□8:00～24:00	□業務時間内	
	2.13	必須		障害からの復旧許容時間（MTTR）		クラスタ構成 バックアップ	稼働率99.9% 稼働時間E：45分以内/月 稼働時間H：40分以内/月 稼働時間M：30分 以内/月 稼働時間L：15分以内/月	稼働率99.5% 稼働時間E：4時間以内/月 稼働時間H：3.5時間以内/月 稼働時間M：2.5時間以内/月 稼働時間L：1.5時間以内/月	稼働率99% 稼働時間E：7.5時間以内/月 稼働時間H：7時間以内/月 稼働時間M：5時間以内/月 稼働時間L：2.5時間以内/月	稼働率98% 稼働時間E：15時間以内/月 稼働時間H：13.5時間以内/月 稼働時間M：10時間以内/月 稼働時間L：5時間以内/月	「リアルタイム・エンタプライズ：事業の継続性と可用性」Gartnerの可用性を元に計算
	－	必須	サイジング参考指標	参照処理のレスポンスタイム		負荷分散	3秒以内	5秒以内	8秒以内	9秒以上	
	－	必須		更新処理のレスポンスタイム		負荷分散	3秒以内	5秒以内	8秒以内	10秒以上	
	5.11	必須	バックアップ	障害時のリストア処理に許容される時間（RTO）		バックアップ	2時間以内	1日以内	3日以内	5日以内	「リアルタイム・エンタプライズ：事業の継続性と可用性」Gartnerより
	5.14	必須		障害時のデータ復旧ポイント（RPO）		バックアップ	直近トランザクション（0時間以内）	4時間以内	1日以内	1日以内	「リアルタイム・エンタプライズ：事業の継続性と可用性」Gartnerより
その他	6.1	必須	システム運用	サーバー設置場所	□データセンター □ビル内サーバールーム □ビル内フロアー	ファシリティ	□データセンター	□データセンター	□データセンター □ビル内サーバールーム	□ビル内フロアー	
	7.1	必須	システム監視	稼働監視の監視レベル	□死活監視 □リソース監視 □プロセス監視 □ジョブ監視 □ログ監視 □監視対象外	運用管理	□死活監視 □リソース監視 □プロセス監視 □ジョブ監視 □ログ監視	□死活監視 □リソース監視 □プロセス監視 □ジョブ監視	□死活監視 □リソース監視 □プロセス監視	□死活監視 □監視対象外	
	7.2	必須		監視体制	□365日24時間　有人監視 □業務時間内　有人監視+夜間オンサイト □オンサイト □担当者による監視	運用管理	□365日24時間　有人監視	□365日24時間　有人監視 □業務時間内　有人監視+夜間オンサイト	□業務時間内　有人監視+夜間オンサイト □オンサイト	□オンサイト □担当者による監視	

システム構成要素	区分	対象要件	Essential	High	Middle	Low
システム（サーバー）	OS/プラットフォーム	利用対象ユーザー	Mainframe Linux/UNIX		Windows Server	
	Webサーバー	システム特性	Apacheベースベンダ製品 Apache	Windows Server		
	外部アクセス制御	システム特性	SSL利用		なし	
	負荷分散	レスポンスタイム	LoadBalancerあり		なし	
	暗号化	利用対象ユーザー	暗号化利用		暗号化利用なし	
	RAID	障害からの復旧許容時間	ストレージ（RAID-10） ストレージ（RAID-5）	ストレージ（RAID-1）	ローカル（RAID-5）	ローカル（RAID-1） ローカル（RAID-0）
	バックアップ対象	障害からの復旧許容時間	全領域 システム領域+データ領域	データ領域のみ	システム領域のみ	バックアップ不要
	バックアップタイミング	サービス稼働時間	オンライン・バックアップ	オフライン・バックアップ		バックアップ不要
	バックアップ時パフォーマンス劣化（オンラインバックアップのみ）	サービス稼働時間	ツールによるパフォーマンス劣化回避	パフォーマンス劣化不可		
	バックアップ方法	サービス稼働時間	フルバックアップ	増分バックアップ 差分バックアップ		バックアップ不要
	スケールアップ/アウト	システム特性	スケールアウト			
APサーバー	OS	サービス稼働時間	Mainframe UNIX	Windows Server		
	データ連携方式	システム特性	EAI/ESB	FTP	ファイル共有	
	H/Wクラスタ構成	サービス稼働時間	Active-Standby（完全Standby型）	Active-Standby（相互Standby型） 負荷分散型	不要	
	RAID	障害からの復旧許容時間	ストレージ（RAID-10） ストレージ（RAID-5）	ストレージ（RAID-1）	ローカル（RAID-5）	ローカル（RAID-1） ローカル（RAID-0）
	バックアップ対象	障害からの復旧許容時間	全領域 システム領域+データ領域	データ領域のみ	システム領域のみ	バックアップ不要
	バックアップタイミング	サービス稼働時間	オンライン・バックアップ	オフライン・バックアップ		バックアップ不要
	バックアップ時パフォーマンス劣化（オンラインバックアップのみ）	サービス稼働時間	ツールによるパフォーマンス劣化回避	パフォーマンス劣化不可		
	バックアップ方法	サービス稼働時間	フルバックアップ	増分バックアップ 差分バックアップ		バックアップ不要
	スケールアップ/アウト	システム特性	スケールアウト			

システム構成要素	区分	対象要件	Essential	High	Middle	Low
DBサーバー	OS	利用対象ユーザー	Mainframe UNIX	Windows Server		
	DB要件	DB製品の選択指針 障害からの復旧許容時間（構成）	DB2 ICE Oracle RAC	Oracle HA SQL Server	Oracle Single	
	負荷分散	レスポンスタイム	DB2 ICE Oracle RAC		なし	
	H/Wクラスタ構成	障害からの復旧許容時間	Active-Standby（完全Standby型）	Active-Standby（相互Standby型） 負荷分散型	不要	
	RAID	障害からの復旧許容時間	ストレージ（RAID-10） ストレージ（RAID-5）	ストレージ（RAID-1）	ローカル（RAID-5）	ローカル（RAID-1） ローカル（RAID-0）
	バックアップ対象	障害からの復旧許容時間 障害時のリストア処理に許容される時間	全領域 システム領域+データ領域	データ領域のみ	システム領域のみ	バックアップ不要
	バックアップタイミング	サービス稼働時間	オンライン・バックアップ	オフライン・バックアップ		バックアップ不要
	バックアップ時パフォーマンス劣化（オンラインバックアップのみ）	サービス稼働時間	ツールによるパフォーマンス劣化回避	パフォーマンス劣化不可		
	バックアップ方法	障害時のデータ復旧ポイント	フルバックアップ	増分バックアップ 差分バックアップ		バックアップ不要
	スケールアップ/アウト	システム特性	スケールアップ			
運用・監視	統合運用・監視サービス	障害からの復旧許容時間	統合運用・監視サービス			利用しない
	システム監視要件	障害からの復旧許容時間	死活監視 リソース監視 ジョブ監視 プロセス監視 ログ監視 データベース監視			
	監視体制	障害からの復旧許容時間	365日24時間　有人監視	業務時間内　有人監視+夜間オンサイト	オンサイト	担当者による監視
	アラート通知	障害からの復旧許容時間	電話による担当者通知	メールによる担当者通知		なし

システム構成要素	区分	対象要件	Essential	High	Middle	Low
バックアップ	統合バックアップサービス	障害からの復旧許容時間	統合バックアップサービス			利用しない
	バックアップ対象		全領域	システム領域+データ領域	データ領域のみ	システム領域のみ バックアップ不要
	バックアップタイミング	サービス稼働時間	オンライン・バックアップ	オフライン・バックアップ		バックアップ不要
	バックアップ時パフォーマンス劣化（オンラインバックアップのみ）	サービス稼働時間	ツールによるパフォーマンス劣化回避	パフォーマンス劣化不可		
	RAID	障害からの復旧許容時間	ストレージ（RAID-10）	ストレージ（RAID-5）	ストレージ（RAID-1）	ローカル（RAID-5） ローカル（RAID-1） ローカル（RAID-0）
	バックアップシステム構成	サービス稼働時間	ミラーディスク切り離し型	スナップショット型 NAS型	ネットワーク経由型	テープ装置直結型 バックアップ不要
	バックアップ保管形態	障害からの復旧許容時間	ディスク装置	テープ装置		バックアップ不要
	バックアップ方法	障害時のデータ復旧ポイント	フルバックアップ	増分バックアップ 差分バックアップ		バックアップ不要
	テープの外部保管	障害からの復旧許容時間	テープの外部保管	テープの外部保管なし		
その他	ファシリティ	サーバー設置場所	外部データセンター			ビル内サーバールーム ビル内フロアー
	開発環境		開発+検証+本番	開発&検証+本番		開発&検証&本番
	開発環境バックアップ		プログラム+マスタ+トランザクション	プログラム+マスタ	プログラムのみ	なし
	H/Wベンダーサポート内容	障害からの復旧許容時間	24時間365日対応		平日9:00～18:00のみ	なし
	H/Wベンダーサポート体制	障害からの復旧許容時間	常駐	オンサイト	センドバック	なし
	S/Wベンダーサポート内容	障害からの復旧許容時間	24時間365日対応		平日9:00～18:00のみ	なし
	S/Wベンダーサポート体制	障害からの復旧許容時間	常駐	オンサイト	センドバック	なし
	WAN冗長化	障害からの復旧許容時間	冗長化あり			
	LAN冗長化（データセンター）	障害からの復旧許容時間	冗長化あり			
	LAN冗長化（バックアップ）	障害からの復旧許容時間	バックアップ専用LAN			冗長化不要
	NIC冗長化	障害からの復旧許容時間	LoadBalancer	Active-Standby		冗長化不要