利用者開放型コンセプトによる航空管制卓デザイン

管制システム部　塩見格一

1. はじめに

航空管制業務を支える情報処理システムは，運航計画やレーダによる監視情報等々の入力情報を，各管制席の利用者に対応させて配信するシステムであり，複数のデータソースからの入力を選択加工して出力するものである点において，何等他のコンピュータのアプリケーションと変わるものではない。特別な点は，社会的に重要な情報を扱っている点と，比較的に長い歴史を有するシステムであるという点であろうが，これらの点は，銀行等のシステムにおいても同様である。

社会基盤としての情報処理システムは，第３世代コンピュータとして 1964 年４月７日に IBM により発表された System/360 により急速に発展し，以降メインフレームを擁するシステムとして今日に至っている。航空管制情報処理システムや銀行の預金管理システム等は，その代表的なシステムであり，またこれらは今日，小型化・低消費電力化を目的として，メインフレームからビジネスサーバへの移行が求められているシステムでもある。

以下，メインフレームからビジネスサーバへの移行に係る問題点について，またビジネスサーバ上に合理的なアプリケーションを構築しようとする場合に考慮すべき事柄について述べる。

また，拡張性が高く保守性に優れたアプリケーションを実現するためのコンセプトとして利用者開放型コンセプトについて紹介し，利用者開放型コンセプトに従えばどれ程に優れた航空管制情報処理システムを実現できるのか明らかにしたい。

2. システム・アーキテクチャ

今日，コンピュータやネットワークの構造や構成を示す言葉としての“アーキテクチャ”という言葉は，先の IBM System/360 と共に使われる様になったものであり，これ以前には明確にはその概念は存在しなかった。System/360 が如何に画期的なものであったかは，半年前のパソコンが旧型に感じられる今日においても管制情報処理システム等として現用されていることからも明らかではある。しかし今日“何でも出来る１台のマシン”を中心としたパラダイムは既に，部分的な目的に合ったマシンを組み合わせて全体としての目的を満足するシステムを実現するものに移っており，分散処理こそが将来的に正しい方向であると考えられていることは誰の目にも明らかではなかろうか。

今日問題とされるアーキテクチャは，コンピュータ１台のアーキテクチャではなく，分散処理を実現するシステムのアーキテクチャである。

航空管制業務の合理化や高度化を実現しようとする場合，問題とすべきが後者のアーキテクチャであることは明らかである。

3. 良いソフトウェアの条件

メインフレームを中心にシステムがデザインされていた当時，コンピュータは極めて高価なものであり，メモリやディスクも，今日では想像もできない程に貴重な資源であった訳であり，必然的に，プログラムはできるだけ少しのメモリで動作する様に製作されていた。

当時，貴重なハードウェアを有効に使うソフトウェアが良いソフトウェアであった訳であり，今日ではファームウェアと呼ばれるオペレーティング・システム等も，その考え方の例外ではない。

例えば FDP において，今日のワークステーション・シェルにおける関数程度の機能が，COBOL で記述された多数のアプリケーションとして実現されていることは，上記の歴史的な経緯による。

しかし今日，コンピュータの価格性能比はメインフレーム時代に比較して数桁以上改善され，システムの整備・運用・保守において要する経費の殆どはソフトウェアに係るものとなっている。**今日，良いソフトウェアの条件は，**最早ハードウェアを徹底的に効率的に利用しようとするものである筈もなく，**明らかに機能変更等の要求に対する保守性が優れていることにこそあると考えられる。**

ハードウェア性能を引き出すことに重点が置か

れたメインフレーム用のオペレーティング・システムやソフトウェアは，最早，今日の良いソフトウェアの条件を満足するものではない。今日，ソフトウェアの開発し易いプラットフォームにおいてのみ，良いソフトウェアは実現可能なのである。

4． データ指向パラダイム

ソフトウェアが “ ハードウェアのおまけ ” であった時代は終わり，今日のビジネスの中心がソフトウェア開発であることは，未だのメインフレーム利用者であっても同様であり，実に“ 同様であること ” こそが，今日，銀行系システムの移行等において発生する様々な問題の根底に存在する。即ち，良いソフトウェアの開発には向かないプラットフォームにおけるソフトウェア開発においては，部外者には説明できそうもない程の，また理解し得ない程の無理が日常的に発生する。

メインフレームに比較して極めて高性能なパソコンが日常的に様々に使用されている現状においては，システム開発者でもない利用者がメインフレームにパソコン並の操作性を要求することは無理のない話であって，その様な要求に応えることが如何に困難であるのか，と言うことについて理解を得ることは殆ど不可能と思われる。

現実に，メインフレームを離れて新たにパソコンやワークステーションを組み合わせたプラットフォームに新たなシステムとして実現するのであれば，最新最高の操作性を有するシステムを実現することは，さほど難しいことではない。

プラットフォームの移行が容易ではないのは，現実の社会における情報処理システムの価値は，その情報処理機能にある訳ではなく，そのシステムの有する情報そのものにあることによる。

旧式なシステムであって，またその情報処理機能は些細なものであっても，長期に渡って社会的な使命を担って使用されてきたシステムには，人間の手作業等によっては決して対応し得ない程の膨大な情報が蓄積されており，その価値こそが，例えばメインフレームからビジネスサーバへのプラットフォームの移行を困難なものとしている。

今後のシステム開発は，そのシステムに蓄積されるデータの長期的な維持が容易であるように，即ち，**将来的なプラットフォームの変更等が蓄積されるデータの価値を何等損なうことのないアーキテクチャを有するものとして行わなければならない。**

5． オブジェクト指向デザイン

コンピュータ黎明期における，その情報処理業務への導入は，従来，人間が手続き的に行ってきた作業の自動化として実現された。人間の行っていた作業の全てを自動化することができた訳ではないため，コンピュータを導入し易い様に，多くの場合，一連の作業の順序を変更する等，業務を再構成してから後にコンピュータの導入が果たされている。このように，一部の手続き的な作業の自動化のためにコンピュータを導入する場合，例えばパンチカードを打つ等，コンピュータに入力するデータを整えるのは人間の仕事であって，また印刷された運航票をホルダーに入れたりする等，出力データを管理するのも人間の仕事となる。

このような用法においては，ソフトウェアの仕様を入力データに対する処理手順の羅列として記述しても，殆どが単純な作業であるから，深刻な問題等が起きることは少なく，アプリケーションの実現には十分であったかも知れない。

しかしながら，ソフトウェアに対する要求が複雑化し，その全体的な機能を個別な機能の羅列として記述することが困難になり，ソフトウェアの仕様を使用目的に対する機能として記述することが必要となった段階では，最早従来手法によってはアプリケーションを実現することはできない。

機能の羅列による記述においては，処理可能な入力は始めから大幅に制限されたものとならざるを得ない訳であって，入力条件の些細な変更にさえ大幅な改修が必要となることは珍しくはない。

複雑な構造を有するソフトウェアを柔軟に実現するためには，システムに実現すべき個々の機能を，それぞれ個別な機能として実現するのではなく，アプリケーション全体を，相互作用の結果として必要な機能を実現するオブジェクトの総体と実現することが必要になる。

個々のオブジェクトは厳密な外部仕様を有するものであり，オブジェクトそのものは従来的な手法によって製作されるソフトウェアと同様に単独では融通の利かないものではあっても，一つ一つを十分に小さく器用に実現しておけば，**その組み合わせは想像以上の柔軟性を実現する**ものとなる。

6. 自律分散型システム

自律分散コンセプトは，1970 年代に，将来的に大規模な情報基盤を実現するためのコンセプトとして提唱されたものであるが，現時点までには，JR 東日本の ATOS 等を例外として，成功を収めているシステムは余り存在しない。[2]

しかしながら，このことは自律分散コンセプトそのものが誤りであったことによるのではなく，集中型のシステムにおいてもかなりの規模の社会的なシステムが実現可能であったことに，また一旦集中型システムとして構築されてしまえば，先に述べた様に“ プラットフォームの変更が困難である ”ことによる。今日においても“ 一個人の理解できる範囲を遥かに超える程に巨大な ”情報基盤の構築を可能と思わせるコンセプトは自律分散コンセプト以外には存在しないと思われる。

自律分散型システムは，場（例えば，データ・フィールド）を共有する無数のサブ・システムから構成され，個々のサブ・システムにおいては，システム機能を実現するための様々なアプリケーションが機能する。

自律分散型システムには各種の実装形態が考えられるが，最も単純には，**「個々のサブ・システムは，常にデータ・フィールドを監視し，処理可能なデータを発見すれば，予め組み込まれた処理を実施し，必要に応じて処理済みのデータをデータ・フィールドに返す。」**といったものを考えることができる。

個々のサブ・システムは「○○なデータが在ったら，○○しよう。」と待ち構えている訳であって，処理のきっかけとなるデータが存在しない限りは，データ・フィールドにおいては何事も起きない。

実は，サブ・システムにこの様な特長を持たせることができることこそが，自律分散型システムの経常的な発展性や，他のコンセプトにおいては実現不可能な程の堅牢性の実現を可能としている。

図１は個々のサブ・システムがデータをデータ・フィールドに放送することとして飛行計画の登録を行うシステムのコンセプトである。航空会社の追加や削除は任意に可能であるし，バックアップ用の FDP については，メインの FDP とは異なるメーカの異なるデータベース管理システムとすることも何等問題なく可能である。

データ・フィールド型の自律分散システムにおいては，データリンク・パフォーマンスの維持が最重要課題となるが，今日のインターネット技術が戦時下における自律分散システムのバックボーンとしての由来を有することからも，星形集中型のシステムの維持運用に比較して特に困難な状況が発生するとは考え難い。

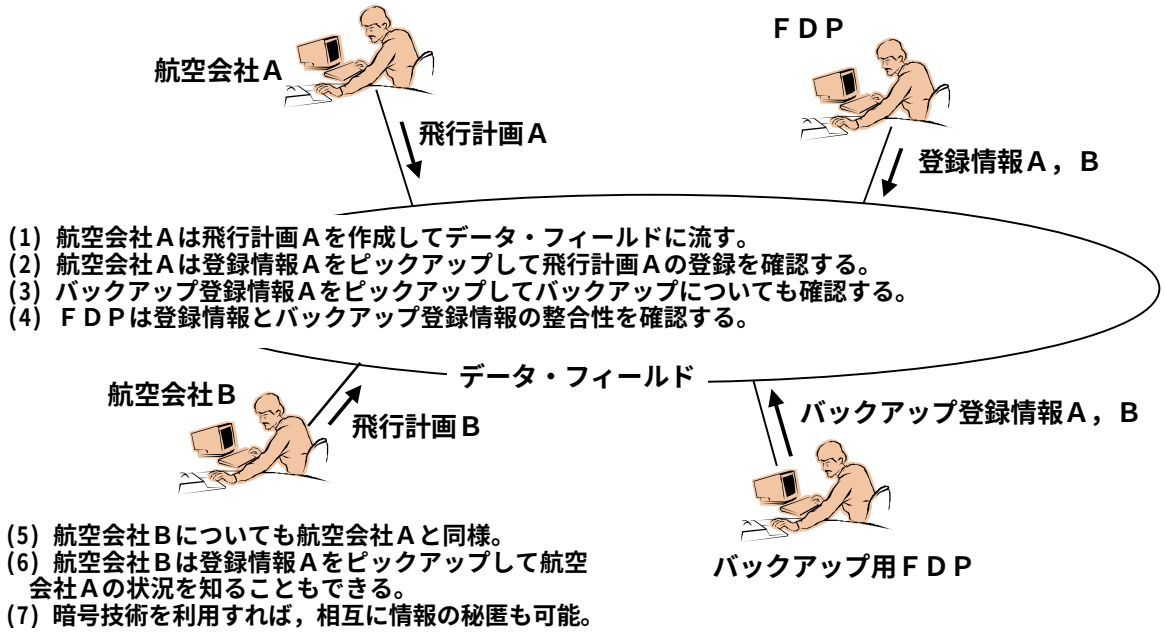

図１　データ・ブロードキャスト型自律分散ＦＤＰ（案）

7. 利用者開放型コンセプト[2]

アプリケーション開発における利用者開放型コンセプトは，必ずしも提唱者の加藤氏の理解とは一致しないかもしれないが，**自律分散型コンセプトを，システムを構成するサブ・システムのレベルから，更にサブ・システム上のアプリケーションを構成するソフトウェア・コンポーネントにまで拡張したものである。**

従来のソフトウェア開発においては，例えば，或るデータベースからデータを取り出そうとすれば，データの抽出条件を設定しこれをデータベースへのメッセージとして送り，その返り値を受け取らなければならならず，また航空機の位置情報をレーダ・イメージとして表示するためには，位置情報を得た後に，更に，そのデータをイメージ描画プログラムに，その入力仕様に対応させて送ること等が必要であり，全体として，目的とする処理が複雑になればなる程に，プロセス間を接続する処理の複雑さが増大する。

利用者開放型コンセプトにおいては，自律的な航空機位置表示プログラム（或いは．アプリケーション）は，データ・フィールド上に“描画すべき航空機の位置情報”を発見すればそれを描画するのみであり，自律的な航空機位置情報管理データベースは，データ・フィールド上にレーダ等センサーからの“航空機位置情報”を発見すればこれを取込み，レーダ・スコープ上の表示を更新する必要のある航空機情報が生成された場合には，“描画すべき航空機の位置情報”としてデータ・フィールド上に流せば良い訳である。航空機の経路別や高度別に処理を分散させることも，航空機位置情報管理データベースの取込んだり流したりする情報を制限するだけで可能であり，表示上の識別等についても，表示プログラムを同時に複数動作させ，それぞれがデータ・フィールドから取込むデータをフィルタリングする等により，任意に描画条件を設定することができる。

今日，アプリケーションのGUIをスライダやボタンを組み合わせて実現するソフトウェア開発環境やそのコンセプトは少しも珍しくはないが，利用者開放型コンセプトは，個々のソフトウェア・コンポーネントが自律的なものであることによって，それらとは全く別なものである。

サイバーラボ社加藤氏の言葉を借りれば，膨大な部品とマニュアルが用意されているからといって誰でもが自動車を作れる訳ではない様に，単に様々な機能要求を満たす部品（ソフトウェア・コンポーネント）を提供しただけでは，監理し切れない複雑さの増大により，ある程度以上の規模のアプリケーション開発は旨くは行かない。

大規模なアプリケーション開発を，またその運用・維持・改修を，何とか破綻させずに進めて行くためには，規模の拡大が複雑さの増大に結び付かない様な開発手法が必要不可欠である。

当所における仮想現実実験施設ソフトウェアや次世代管制卓評価システムの開発等において，CORBAやHLA等によりアプリケーション間の接続仕様を明確化することも確かに有効ではあったが，3つ以上のソフトウェアを相互接続しようとする場合や，ブラックボックスをCORBA等によりレガシー・ラッピングしようとする場合等，従来手法によっては不可能であったかもしれないが，CORBAによっても，それ単独では必ずしも容易に目的が果たせた訳ではない。僅かな複雑さではあっても，相互に接続しなければならないアプリケーションやモジュールが増えれば，その組み合わせによる複雑さの増大は途方もないものになってしまうことが理解された。

大規模なソフトウェアの開発においては，細かな部品同士の関連を利用者が知らなくとも自動的に部品が組み上がる仕組みが実現されていることこそが必要なのであって，この仕組みを全てのオブジェクトのスーパークラスに記述実現し，その性格をアプリケーションを構成する全てのオブジェクトに継承させ持たせることこそがプロジェクト成功の鍵となる訳である。

アプリケーション開発における利用者開放型コンセプトの意義は上記の様に極めて大きいが，更に，自律分散型のコンポーネントにより組み立てられたシステムは，任意のコンポーネントの増減が可能であり，その性格から，堅牢性の向上のみならず，任意に人間がシステムの処理に割り込むことを可能とする構造を容易に実現し得るものとなっている。

［参考文献等］

[1] http://www.curiosat.jpn.org/
[2] http://www.cyberlab.co.jp/