マルチベンダ・インテグレーション・アーキテクチャに基づくアプリケーションの移植性評価

3D－3

山田　竜二　　仲谷　元
ＮＴＴ　ソフトウェア本部

１．はじめに

ＮＴＴでは、ベンダ・フリーなシステム構築を目指し、ＡＰの移植性、相互接続性を保証するためのＭＩＡ（Multivendor Integration Architechture）仕様を規定した。我々はＭＩＡを適用した約３００Ｋｓの業務共通アプリケーション（以下、ＣＡＰと呼ぶ）を開発し、料金計算、電話番号案内等のＮＴＴのオンライン・システムで稼動させるとともに、メインフレーム４ベンダ環境間での移植を実施し、ＭＩＡを適用したアプリケーションの移植性、並びに性能の検証を実施した。本稿では、この移植実績に基づき、ＭＩＡを適用したアプリケーションの高い移植性と性能の確保のために考慮すべき事項について報告する。

２．ＭＩＡ仕様のベンダ依存性

ＭＩＡ仕様では国際標準言語（COBOL、C、SQL）を元に必要に応じて機能の追加／削除を行い、実装依存とされている項目についても可能な限り規定した。更に、標準化の進んでいないトランザクション制御言語として、ＳＴＤＬ（Structured Transaction Definition Language）を規定し、アプリケーションの移植性を確保している。ただし、ＭＩＡの仕様化に当たっては、既存ベンダ製品との整合性を確保し、数多くのベンダが容易に実装できるようにとの配慮から、一部にベンダ実装固有な仕様を許している。

３．移植対象ＡＰの特徴

移植を行ったＣＡＰは以下の特徴を持つ。

①ＮＴＴのオンラインシステムで共通的に必要な制御機能（ファイル転送機能、スケジュール機能、業務ＡＰ起動チェック機能等）を実現している。

②ＳＴＤＬ並びにＣＯＢＯＬで記述

４．移植性向上のための考慮点

ＣＡＰの移植性向上のために以下の点を考慮て開発した。

①ＭＩＡが規定している（各ベンダがサポートしなければならない）最小アーキテクチャ定数の範囲内で開発した。

②ベンダが実装時の都合により制限できる項目の範囲内で開発した。

③ＭＩＡで実装固有としている言語部分は極力使用せず、ポータブルな言語の範囲内で記述（ポータブル部）した。

④ＭＩＡ規定外機能の局所化（ベンダ依存部）を行い、ポータブル部とのインタフェースを規定し、ベンダ依存部の調達時要件とした。

⑤ＭＩＡ仕様が正しく反映されず、ベンダ固有の記述がされることを防ぐため、開発段階でポータビリティ・チェック・ツールによるチェックを行った。

⑥重複する可能性のある各種資源の名称付与基準を規定し、システム個別のＡＰを含めて今後のＡＰ開発においてもこの規定が反映されるのものとした。

５．移植の流れ

複数の開発ベンダ環境から複数のターゲットベンダ環境へＡＰ移植を行う場合、各開発ベンダ環境から全てのターゲット・ベンダ環境対応の修正を行わなければならず（図１）、ターゲット・ベンダが増えると移植やメンテナンスのために膨大な稼働を要する。

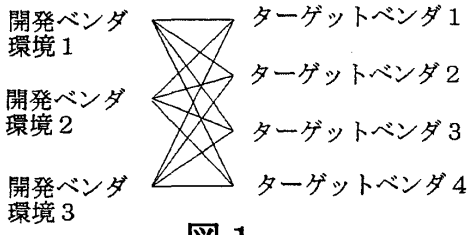

図１

この問題を解決するために、機能毎に各開発ベンダ環境で開発されたＣＡＰソース・プログラムの標準化（ベンダ依存な記述方法に関して、各ベンダの実態調査を行い、一番多い記述方法に合わせて修正すること）を行い、それを各ターゲット・ベンダへ移植した（図２）。

---

* Evaluation of the AP portability based on Multivendor Integration Architecture
Ryuji Yamada　Hajime Nakatani
NTT Software Headquarters

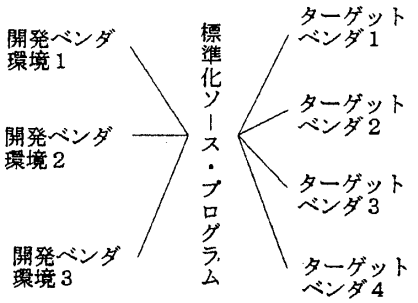

図２

６．移植性検証

標準化ソース・プログラムを各ターゲット・ベンダ環境へ移植した結果を表１に示す。

表１　移植結果

ターゲット・ベンダ	ＳＴＤＬの移植率	ＣＯＢＯＬの移植率
Ａベンダ	１００％	１００％
Ｂベンダ	１００％	９８％
Ｃベンダ	９９％	９８％
Ｄベンダ	９９％	９９％

＊移植率＝（総ステップ数－修正ステップ数）÷総ステップ数

修正が必要な項目は以下のように分類することができた。

ＳＴＤＬ：

①ＭＩＡ仕様の曖昧さが原因（２項目）

②ＭＩＡ仕様実装上の差異が原因（２項目）

ＣＯＢＯＬ：

①言語仕様上の差異が原因（２項目）

②ＯＳの差異が原因（２項目）

③ＭＩＡ仕様実装上の差異が原因（１項目）

７．性能検証

ＭＩＡを適用したＡＰはトランザクションの開始／終了等の制御をＳＴＤＬで記述することになる。一方、新たなトランザクションが起動されると、呼び元と呼ばれる側は別のジョブで実行されるため、ジョブ間のリンクやインターフェースの整合性チェックのために環境情報の参照を行う。環境情報はメモリ上に実装されている場合もあるが、ファイル上に実装されている場合もある。

このため、ＳＴＤＬによるトランザクション制御部分のネスト構造が複雑になったり、やたらとトランザクション分割を行うと、性能の悪化が予想される。

実際に７つのネストで実現された機能（約１．８Ｋｓ）を１つのネストで実現し、双方の性能実測を行ったところ、前者に比べて後者の実行時間は２５％～５０％であった。

８．考察

（１）移植性

今回の移植結果から、オンラインＡＰを開発する上でベンダ依存項目の全てが必要である訳ではなく、１０６項目中７項目のみで、非常に移植率の高いオンラインＡＰを開発可能であることが判った。また、修正が必要なそれぞれの項目についても以下の対策により解決できる。

・ＳＴＤＬ－①に対する対策
　仕様バグとして修正

・ＳＴＤＬ－②、ＣＯＢＯＬ－③に対する対策
　標準化ソース・プログラムの修正

・ＣＯＢＯＬ－①、②に対する対策
　修正の自動化

これらのことから、移植は全て自動修正でき、要する稼働もわずかとなる事が予想される。

（２）性能

トランザクション制御部分のネスト構造を簡略化することにより性能がはるかに向上することが判ったが、以下のような問題も存在する。

①同一の処理が複数存在することにより、メンテナンスが煩雑になる。

②共通モジュールによる、機能間でのモジュール共用ができない。

③メモリ、ファイル等の資源占有時間が長くなる。

実際にシステムを開発する場合は、性能への影響を評価の上、ＡＰのモジュール構成の検討を行っていく必要がある。

９．終わりに

ＣＡＰは一般の業務ＡＰに比べ制御系よりのソフトである。ＣＡＰの開発を通じてＭＩＡ仕様の高い移植性が実用面でも実証されたと考える。

ＭＩＡ仕様は、その後世界のキャリアと共同して標準化を進めたＳＰＩＲＩＴ（Service Providers' Integrated Requirements for Information Technology）仕様のベース仕様となり、Ｘ／ＯＰＥＮ仕様として採用され、国際標準化を果たした。今後、ベンダ・フリーなプラットフォーム仕様として広く普及していくものと考える。