ソフトウェア設計の革新
Architecture-oriented Software Design

深谷　哲司 FUKAYA Tetsuji
吉田　和樹 YOSHIDA Kazuki
細谷　竜一 HOSOYA Ryuichi
宮田　尚志 MIYATA Takashi
玉木　裕二 TAMAKI Yuji

機能要求の高度化や実行プラットフォームの多様化など，ソフトウェアシステムを取り巻く環境は変化し続けている。これらの変化のなかでタイムリーに製品を提供しビジネスチャンスを拡大するには，ソフトウェアの開発も，変化に柔軟かつ俊敏に適応するものでなければならない。当社では，ソフトウェア アーキテクチャの確立がそのようなソフトウェア作りで中心的な役割を果たすと位置づけ，アーキテクチャ中心のソフトウェア設計技術の確立・普及を推進している。まず，各製品分野の特性を反映したアーキテクチャを構築し，次にそのアーキテクチャを基にしたソフトウェアのひな型となるフレームワークを準備し，最後にフレームワークに基づいてソフトウェアを開発する，というアプローチである。この技術を適用した製品ドメインでは，従来の開発と比較して30 %以上のQCD（Quality, Cost, Delivery）改善を実現している。

The infrastructure and requirements of software systems are rapidly changing nowadays. Flexibility and timeliness are key factors for grasping as many business opportunities as possible. Under such circumstances, software system development requires flexible and agile adaptability to various changes. These papers describe an architecture-oriented software development technology which is implemented in the following steps: (1) construction of the software architecture reflecting the characteristics of each product domain; (2) preparation of a framework as an archetype of the target software based on the architecture; and (3) development of the target software using the derived framework. Using this technology, we have improved the quality, cost, and delivery (QCD) of a certain product by 30 % compared with the conventional development method.

ソフトウェア設計への取組み──概論
Overview of Software Design Approach

1 まえがき

現在のソフトウェア開発には，新しいサービスの迅速な提供やビジネスモデルの変化への俊敏な対応が求められており，既存ソフトウェアやその開発過程に関する資産の活用が必須である。ここでは，既存ソフトウェア資産の効果的，効率的な活用を実現する目的で当社が開発・普及を推進しているアーキテクチャ指向開発ACE（Architecture Centric Engineering）手法の概略について述べる。

ソフトウェア アーキテクチャとは，ソフトウェアの全体構造とそれぞれの部分を構成する要素間の関係を表現するものである。大規模化，複雑化の一途をたどるソフトウェアシステムの開発における既存資産の有効活用では，ソフトウェアアーキテクチャが中心的な役割を担う。最適なアーキテクチャは，ソフトウェアシステムの製品分野やその製品系列の開発戦略によって異なるが，その設定・選択は非常に重要で，将来の製品開発計画や製品を取り巻く状況までも考慮して決定されなければならない。誤ったアーキテクチャは，その後の開発での大きな後戻りや品質の低下に直結する。

2 ソフトウェア設計の課題

既にあるソフトウェア資産を効率的に活用するための様々な技術が提案されている。しかしその一方で，多くの再利用技術では，次のような問題が指摘されている。

⑴ ソフトウェアコンポーネントなどの再利用の限界

再利用性の向上のために，オブジェクト指向開発，コンポーネント指向開発が普及している。しかし，コンポーネントなどの再利用部品が用意されていても，アーキテクチャ上の不整合（architectural conflict）により利用できない問題が表面化している。ある部品を利用するには，その部品が想定しているアーキテクチャ上の約束事を守る必要がある。この約束が明確に認識，統一されていないと，再利用はスムーズに進まない。特に，粒度の大きな部品を再利用する場合に，この現象は顕著である。

⑵ アーキテクチャ不在の弊害　初期の設計方針として明確なアーキテクチャが定められていないと，モノリシックな構造に陥る場合が多い。その保守・拡張は難しく，たとえ局所的な変更であっても大規模な作り直しが必要となる。また，大規模ソフトウェアでは，わずかな機能変更が致命的な品質低下の原因になる場合もある。

3 アーキテクチャ中心のソフトウェア開発

ACEに基づくアーキテクチャ指向開発とは，ソフトウェアの最適なアーキテクチャを開発の初期段階で十分に検討し，そのアーキテクチャに沿った形で整然と開発を進めることであり，オブジェクト指向やコンポーネント指向開発におけるソフトウェア資産再利用の効果を最大化することを目標とするアプローチである。

3.1 非機能要求を考慮した上流分析とアーキテクチャ設計

ソフトウェアに対する要求には，それが提供するサービス機能そのものに関するものと，機能そのものではなくソフトウェアの保守，移植や拡張性などに関するものとがある。ここでは，前者を機能要求，後者を非機能要求と呼ぶ。ACEは，開発の初期段階から機能要求だけではなく非機能的要求も特に考慮して開発を進めることを特徴としている。つまり，開発の初期段階に非機能要求を，アーキテクチャを特徴づける特性項目として定義し，これら特性項目の検討を十分に行ったうえで最適なアーキテクチャを構築する。

アーキテクチャ特性の決定については，ビジネス戦略も考慮する。製品系列をどのように開発するか，どの特性が製品として重要であるかを見極め，優先度付けされた要件として整理する。特性によっては相反する要件もあるため，体系的に比較検討し全体の方針を決定する。例えば，再利用性に関しても，製品系列上のどの範囲をどのように再利用するかを明確に決定しておく。

アーキテクチャそのものの設計は，上記のアーキテクチャ特性を考慮して，個別に最適化を行う過程である。特性項目の評価を基に，既に事例として蓄えられたアーキテクチャパターンの中から最適なものを選択することも可能である。

特に，再利用性，拡張性を重視したシリーズ製品の開発では，構築されたアーキテクチャを基にしてフレームワークを整備する。フレームワークとはシステムのひな型であり，アーキテクチャで定められた制御方式やインタフェースがあらかじめ半製品として提供される。フレームワークを利用すれば，シリーズ内の各製品は，シリーズからの拡張範囲を差分として開発するだけで完成する。

3.2 適用範囲

アーキテクチャ指向開発のアプローチは，適用対象となる製品分野を限定しない。しかし，このアプローチを利用して構築されるアーキテクチャは対象製品やそのビジネス戦略により大きく異なる。現在，ACEのアプローチは当社が扱うソフトウェアの大きな領域である業務システムと組込み制御ソフトウェアをカバーしている。

3.2.1 業務システム（ビジネスアプリケーション）

企業内の会計システムなどビジネス諸業務を扱うシステムの開発には次のような課題がある。

⑴ 新しいIT（情報技術）を短期間で導入しなければならない。

⑵ 複雑な業務内容に短期間で精通しなければならない。

これらの課題を克服し，短期間でのシステム構築を実現するために，業務システム向けアーキテクチャを策定し，これに基づきAPF（APplication Framework）を用意している。また，このアーキテクチャとフレームワークにより保守性と品質の向上を達成している。

3.2.2 組込みシステム

機器制御や情報家電などの組込みシステム分野のソフトウェアには，安全性，応答性などの厳しい制約がある。また，頻発する制御機器（デバイス）などの変更要求にも迅速に対応しなければならない。これらの制約を満足するために，次のようなアーキテクチャやフレームワークを構築し，開発期間の短縮と品質向上を達成している。

⑴ リアルタイム処理を容易に実現できる。

⑵ 制御デバイスの多種多様化に迅速に対応する。

4 あとがき

製品要求や技術の変化に対応したタイムリーな製品提供を可能にするソフトウェア開発技術のニーズはますます高まっている。ACEのアプローチを多くの開発に適用すると同時に，ベストプラクティスを再利用できる形式にまとめ，よりいっそうの生産性向上に注力していきたい。 （深谷）

業務システム向けアプリケーションフレームワーク

Application Framework for Business System

1 まえがき

企業内の情報などを扱う業務システムは，目まぐるしく進歩する様々なITを駆使して開発される。システム開発者は，これらの技術をいち早く導入して，品質の高いシステムを効率よく開発しなければならない。また，業務システムの開発には，対象業務の理解が不可欠である。システム化を進める業務やビジネスモデルを正確に理解しないと，適切なシステムを構築することは難しい。このように，最新のITや複雑な顧客のビジネスモデルが，業務システムの開発を難しいものにしている。

当社は，これらの課題を解決するために，業務システムのための開発体系APFの開発と適用を進めている。APFは，①統一されたアーキテクチャ，②アーキテクチャを構成する業務コンポーネントを構築するための基本コンポーネントセットと開発環境，③APFによる開発をガイドする開発方法

論，から構成される。ここでは，この中で特に，アーキテクチャ，基本コンポーネントセット，開発環境を説明し，また適用事例を通して，APFに基づく開発の概要について述べる。

2 業務システムアーキテクチャ

2.1 APFの基本アーキテクチャ

一般に業務システムは，機能的に，次のように分解できる。

⑴ 業務／処理フローや画面の遷移を扱う処理群
⑵ 上記には依存しない独立した処理群
⑶ データベース（DB）処理に関連する処理群
⑷ 業務遂行上のルールなどに関する処理群

システムの実装やその後の拡張・保守を考えると，ソフトウェアアーキテクチャは，まず⑴と⑵を明確に分離して構成することが望ましい。DB設計に対する要求は多種多様で変更も多く発生するため，⑶のDB関連処理も他の処理から分離して扱えることが望ましい。更に，⑷の業務ルールに関する機能は，システムのライフサイクルを通じて，あるいは他システムへの転用を考えるうえで，可変的に扱える必要がある。

APFでは，これらを考慮して，“業務プロセス”，“業務サービス”，“業務エンティティ”，“業務ルール”を明確に区別した業務システムアーキテクチャを採用しており，ビジネス業務の一貫した分析・設計作業を可能にしている。また，設計結果から，指定された実行環境で動作するコードを自動生成する機能を提供することで，ITの変化がソフトウェア実装に与える影響を最小限にとどめるようになっている（図1）。

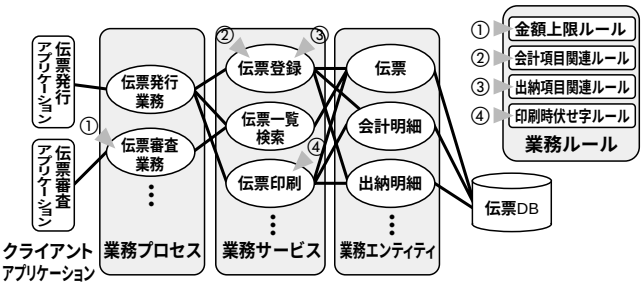

図1．業務システムのためのアーキテクチャ　“業務プロセス”，“業務サービス”，“業務エンティティ”，“業務ルール”から構成される。
Software architecture for business application based on application framework (APF)

2.2 アーキテクチャの実現方式

APFは，このアーキテクチャに基づくシステムの開発を，コンポーネントの階層的な組合せにより実現する。つまり，前述の処理群を個々のアーキテクチャレイヤに振り分け，そのうえで，各レイヤで実現すべき処理を業務コンポーネントとして開発し，その組合せでシステム全体を構築する。APFは，この業務コンポーネントを開発するための基本コンポーネントのセットも提供している。ITを扱うために特化された処理などは，この基本コンポーネントの中に隠蔽（いんぺい）されて実装されている。

このようなコンポーネントベースの開発スタイルを採用することにより，システムの品質，保守性は向上し，システムに対する要求の変更にも俊敏に対応できるようになる。

3 基本コンポーネント

フレームワークは，基本コンポーネントのセットとそれらの組合せや各レイヤ間を連携する機構を備えている。

ここでは，“業務プロセス”のコンポーネント開発に利用される基本コンポーネントのセット“フロー制御”と，業務サービスのコンポーネント開発に利用される基本コンポーネントのセット“データ管理”，また，これらの基本コンポーネントを組み合わせる際に利用する開発環境の概略について述べる。

3.1 “フロー制御”コンポーネントセット

アプリケーション内の処理の流れを制御し，後述するデータ管理で作られる機能を呼び出したり，画面との同期を取ったりするための基本コンポーネントのセットである（図2）。

3.2 “データ管理”コンポーネントセット

DBに登録されるデータを管理し，それに伴う業務ロジックとデータの加工処理を実行するための基本コンポーネントのセットである。DBへのアクセス処理を行う業務エンティティ コンポーネントは，データ管理とともに提供している“リレーショナルDBラッピング”モジュールから生成される。このモジュールは，データ定義とデータ操作を表現したスキーマから登録，検索，更新，削除などの基本機能を備えたコンポーネントを自動生成することができる（図3）。

3.3 開発環境

APFが提供する開発環境では，基本コンポーネントをビジュアルに組み合わせることで，アプリケーションを分析し，

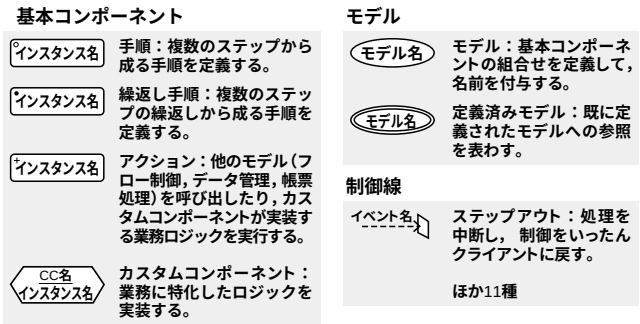

図2．基本コンポーネントセット―フロー制御　アプリケーション内の処理（プロセス）の流れを制御し，後述するデータ管理や帳票処理で作られる機能を呼び出したり，画面との同期をとるためのコンポーネントセットである。
Set of base components, etc., for flow control

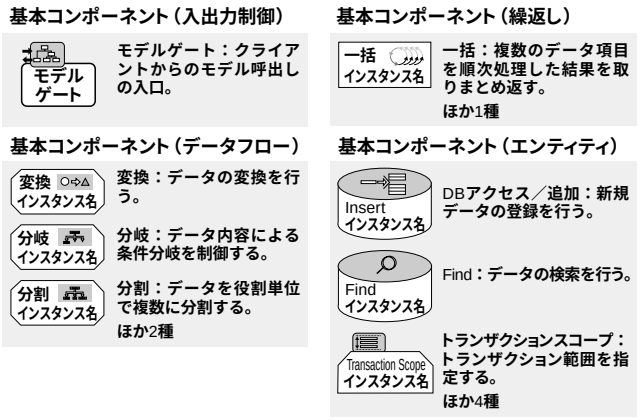

図3．基本コンポーネントセット―データ管理　DB上でのデータを管理（登録，検索，更新，削除）し，それに伴う業務ロジックとデータ加工処理を実行するためのコンポーネントセットである。
Set of base components for data management

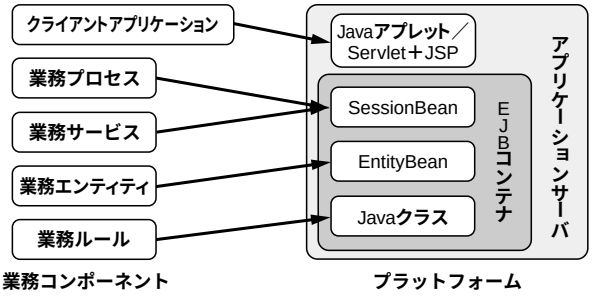

図4．コンポーネント自動生成機能（EJBの場合）　業務プロセスコンポーネントと業務サービスコンポーネントはSessionBeanに，業務エンティティコンポーネントはEntityBeanに，それぞれマッピングされる。
Automatic generation of component in case of EnterpriseJavaBeans (EJB) platform

設計することができる。APFは，この設計情報を基に，指定された実行環境で動作する業務コンポーネントを自動生成する。実行環境は，システムの規模，機能要件，性能要件などの各種条件によって変わる。APFは，EnterpriseJava Beans（EJB）[注1]などのあらかじめ想定される業界の標準的な実行環境に対応したコンポーネントの自動生成機能を複数用意することで，実行環境間での差異を吸収している（図4）。

4　適用事例 ― 施設予約管理システム

4.1　概要

施設予約管理システムは，公共や民間施設の利用希望者が，利用したい施設や備品をWebブラウザを通して予約，変更，確認するためのシステムである。また，施設管理者による施設・備品情報や，ユーザーの情報の変更や，施設別の利用統計情報の作成機能を備えている。ここでは，新規予約登録機能開発にAPFを適用した例について述べる。

4.2 “フロー制御”コンポーネントを利用した設計

新規予約登録機能では，利用したい施設や備品の空き状況を確認して，予約に必要な情報を入力することで予約を行う。この業務の流れをフロー制御の基本コンポーネントで構成すると図5のようになる。この基本コンポーネントの組合せ全体が一つの業務コンポーネントになる。

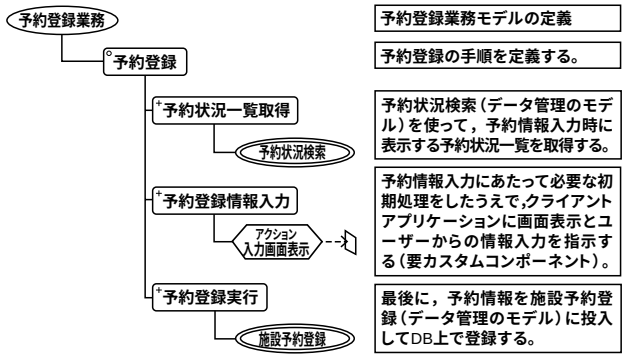

図5．新規予約登録機能へのフロー制御の適用　新規予約登録機能の業務プロセスをフロー制御の基本コンポーネントセットで構成する。
Application of flow control to construction of reservation function

4.3 “データ管理”コンポーネントを利用した設計

図5に示した“施設予約登録”は業務サービスのコンポーネントとして実装される。このコンポーネントは，データ管理の基本コンポーネントを使い，図6のように構成される。

このように一連の処理を，アーキテクチャに基づき，APF

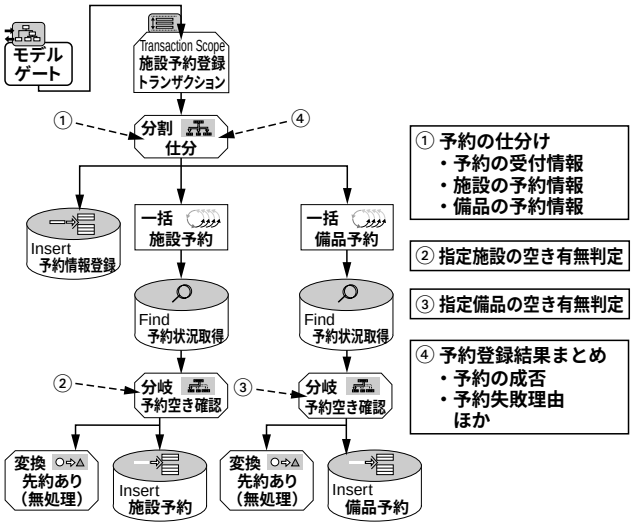

図6．施設予約登録への“データ管理”の適用　施設予約登録の業務サービスをデータ管理の基本コンポーネントセットで構成する。
Application of data management to construction of facility reservation function

（注1）　Java及びその他のJavaを含む商標は，米国SunMicrosystems社の商標。

が提供する基本コンポーネントの組合せと，それにより構成される業務コンポーネントの組合せで，階層的に実装する。

5 APFの適用効果

5.1 品質向上

⑴ 高品質のコンポーネント利用による品質の安定化
汎用的な処理部分，及び，IT技術に関連する処理部分は，高品質の基本コンポーネントで置き換えることになるため，バグ混入の機会が減る。

⑵ 設計が単純化され，不具合混入の削減が可能
処理をコンポーネントの組合せにより設計するため，アプリケーション全体でクラス数が減少する。このため，設計を単純化し，設計品質のばらつきを抑えられる。

5.2 保守性向上

⑴ 設計レベルでの保守が容易　APFにより自動生成されるソフトウェアの保守は，基本コンポーネントの組合せとして表現された設計情報の保守によって実現できる。また，必要に応じて，基本コンポーネントを変更し，他のコンポーネントには影響を与えずに結合することも可能で，保守性の高いソフトウェアを提供できる。

⑵ エラー原因の早期究明が可能　APFは，個々のシステム開発での作り込みが難しい高度なエラー処理ロジックや実行ログ機能を提供している。これらを利用することで，エラー発生時に，「業務コンポーネント内のどの部分でどのような問題が発生したか」などの情報が得られるため，原因の早期究明が可能になる。

6 あとがき

ここでは，業務システムのための開発体系APFを紹介した。今後，APFに従った業務コンポーネントの品ぞろえを増やすことを考えている。これにより，当社内で開発する業務システムの品質と保守性の向上を達成する。（吉田／細谷）

組込みソフトウェア開発向けフレームワーク

Framework for Embedded Software

1 まえがき

家電製品や産業用機械などに組み込まれて，これらを制御する組込みソフトウェア（以下，組込みソフトと略記）は，近年大規模，複雑化してきており，かつ開発サイクルも短くなってきている。

パソコン（PC）／エンジニアリングワークステーション（EWS）上のソフトウェアと異なり，組込みソフトのアーキテクチャは，実行効率やメモリなどの厳しい制約を考慮したうえで，ソフトウェアの柔軟性や再利用性を実現しなければならない。

当社では，組込みソフトの柔軟性や再利用性の向上を目的に，ACEを応用してオブジェクト指向（以下，OOと略記）技術に基づくアーキテクチャの構築と実装の両方を支援するフレームワーク，MTF（Multi Task Framework）/DDF（Device Driver Framework）[1]を構築した。

ここでは，MTF/DDFとともに，その適用事例として，当社の顧客である富士ゼロックス（株）と共同で行ったプリンタ制御ソフトウェア開発について述べる。

2 組込みソフト開発の課題

一般的な組込みソフトにはシステム要件からくる次のような特徴がある。

⑴ リアルタイム性（決められた時間内で応答すること）を確保するため，複数の処理を並行に行う（以下，“並行性”と呼ぶ）必要がある。

⑵ PC/EWS向けのソフトウェアとは異なり，多種多様なハードウェアを制御する必要がある（製品コストなどの関係から同じ製品群でもハードウェアが変更される）。

OO技術はこのような特徴を持つシステムの開発に有効なアプローチである。しかし，OO設計から実装へ進むには，次のような問題がある。

⑴ 特に，OOプログラミング言語であるC++言語を用いる場合には，C++言語に並行性を扱う言語仕様や枠組みが存在しない。

⑵ 製品分野に依存するソフトウェア部分とハードウェアを制御する部分（以下，“デバイスドライバ”と呼ぶ）を切り分けることが難しい。更に切り分けたデバイスドライバを実装するには高度な技術が必要である。

上記の問題の解決には次の方法が考えられる。

⑴ 組込みソフト全般で必要とされる並行性やデバイスドライバを扱うアーキテクチャを規定する。

⑵ 個々の組込みソフトの並行性やデバイスドライバの実装を容易にする手段を提供する。

当社では，上記二点を達成することを目的に，組込みソフト全般で活用可能なフレームワークMTF/DDFを開発した。

3 組込みソフト開発向けOOソリューション

3.1 組込みソフト向けOOフレームワーク

前節で述べた並行性とデバイスドライバの部分は多くの応用で共通して使用するインフラストラクチャ（以下，インフラと略記）部分である。従来の組込みシステムの開発では

PC/EWSの場合に比べ，インフラ部と各製品に依存するソフトウェア部分（以下，アプリ部と略記）との境界が不明確，又は界面のレベルが不均一であることが多かったが，インフラ部分とアプリ部分を体系的に分割することが重要である。

そこで，二つの部分を明確に分離し，インフラ部では当社MPUの性能を引き出すためのノウハウを再利用可能な部品として提供することで，アプリ部との最適な組合せ方法を規定するフレームワークを構築した。それがMTF/DDFである。

⑴ MTF　OO設計モデルのリアルタイムOS（基本ソフトウェア）（RTOS）上への展開を容易にすることを目的としたフレームワークで，並行プログラムの再利用性向上を図る。

⑵ DDF　アプリ部に最適なデバイスドライバのインタフェース構築，及びデバイスの迅速な変更を可能にすることを目的としたフレームワークで，当社MPU内蔵の周辺回路に対するソフトウェア部品を提供する。

3.2 フレームワーク利用のサポート

このようなフレームワークの提供だけで，すぐに組込みソフトOO開発ができるとはかぎらない。OO開発を推進するには，開発者のスキル，モチベーション，開発プロセスなども重要な要因である。また，効率の良いシステムの構築には，製品分野に固有のノウハウを活用する必要がある。このため，当社ではMTF/DDFの提供と併せて，インフラ部のノウハウと顧客の製品分野のノウハウを共有するための共同開発（協業）サポートを行っている。

4 開発事例

当社は富士ゼロックス（株）と共同で，プリンタ制御ソフトウェアのOO技術を用いた開発を行い，MTF/DDFを適用した。ここではその適用アプローチと成果を述べる。

4.1 開発背景と問題点

富士ゼロックス（株）は，従来からμITRON（μ Industrial The Realtime Operating system Nucleus）仕様OS[(2)]と独自開発のシミュレーション環境を導入してプリンタ制御ソフトウェア開発の効率化を図っている。今回，制御ソフトウェア開発のよりいっそうの効率化を目指して，OO技術導入を協業の形で推進した。OO技術導入は下記を目標としている。

⑴ 初期の実機は機械部品／電子部品とも信頼性が確認されておらず，トラブルの切分けが難しい。実機の開発状況に影響を受けないソフトウェア開発を実現する。

⑵ ノウハウが必要となるデバイスドライバ部分を共通ソフトウェア部品として整備する。

更に，プロジェクト運営上の目標として次の点を加えた。

⑶ OO技術の初期導入オーバヘッドを抑える。

4.2 開発アプローチ

これらの目標のうち，⑴はMTF/DDFのほかに仮想ドライバ，RTOSを提供することによって，⑵はMTF/DDFを用いることによって達成した。また，目標⑵及び⑶の達成に向けて図7に示すような段階的な開発環境を構築した。

ここで段階的な開発環境とは，開発の初期は図7左側のようにWindows®(注2)上にMTF/DDFを移植し，そこで上位のアプリケーション部を開発し，中期以降は，図7右側のように，より実機に近い環境を当社製RISC（縮小命令セットコンピュータ）マイコン TX System RISC TX19シリーズ 評価ボード上に構築し，システム全体の整合性を考慮した開発を進めるものである（図8）。段階的開発環境によって，⑵に対してはDDFを活用することにより，デバイスに依存しない実機レスでの開発ができ，また⑶に対しても，MTFを活用することにより，μITRON仕様OSを利用して開発されてきたソフトウェア資産を生かしつつ，段階的にアプリケーション

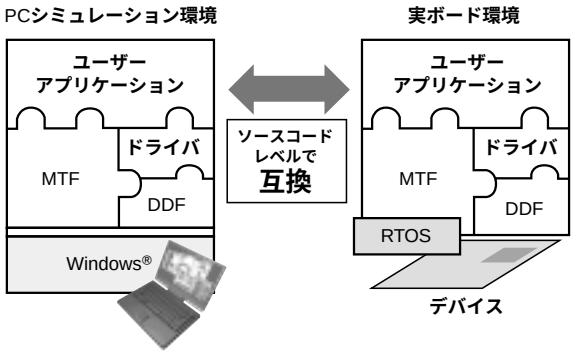

図7．段階的開発環境におけるソフトウェア構成　ユーザーアプリケーションは，再ビルドだけで両方の実行環境へ移行することができる。
Software structure in development environment

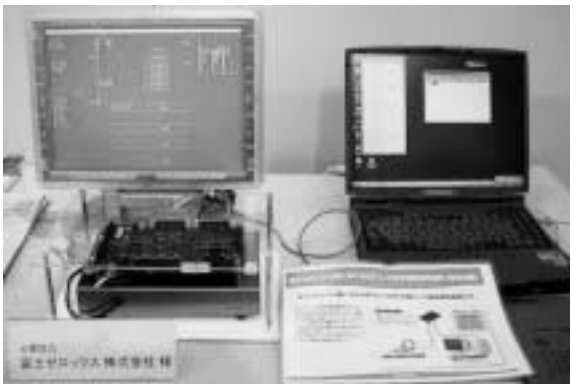

図8．TX19評価ボードを使用した開発環境　左下が評価ボード，左上のPCがメカシミュレータである。
Development environment using TX19 reference board

（注2）　Windowsは，米国Microsoft Corporationの米国及びその他の国における登録商標。

の再構築ができる。これら開発環境の移行に伴う変更は，MTF/DDFで吸収することができ，アプリケーションはソースコードレベルで互換性を保つことが可能である。

4.3 適用結果

富士ゼロックス(株)において今回のアプローチに関する評価が行われた。以下では，その概略を述べる。

⑴ 開発コスト削減　まず，MTF/DDFが提供するソフトウェア部品群を利用し，それらが規定する作成方針に従ってソフトウェアを開発することにより，ソフトウェア作成効率が非適用部に比べて46％向上した。また，シミュレーション環境を利用することにより，評価に用いる実機台数を削減することができたため，ハードウェアの試作回数を減らすことができた。

⑵ 開発期間の短縮　開発期間を32か月から20か月へ短縮することができた。これは，シミュレータによる効率的な不具合検出の効果が大きいと考えられる。ボードでの開発が主体となる開発工程の後半においても，問題切分けのためにシミュレータが効果的であったと考えられる。**図9**に示すように，不具合の修正時間もMTF/DDFを用いた今回のほうが，開発後期でより短縮されていることが確認できる。

⑶ 品質向上　MTF/DDF適用部の発生不具合件数が非適用部に比べ，30％減少している。その理由としては，MTF/DDFが提供する検証されたソフトウェア部品群を利用したことが挙げられる。

⑷ ROM/RAM容量削減　C++言語を使用したソフトウェアのROM/RAM容量は一般的に増加すると言われている。しかし，この開発においてはMTF/DDF非適用部分に比べ，ソースコード1行当たりのROM容量が減少している。これは，MTF/DDFと再設計によりアーキテクチャが統一されたため，複数のプログラマーが重複して作成する部分がなくなったためと推察される。RAM容量についてもモジュール分割単位をクラスとし，必要最小限のタスクを用いて並行性を実現したことにより，減少したタスクのスタック分が減少している。適用前に比べ30％削減されたモジュールもあった。

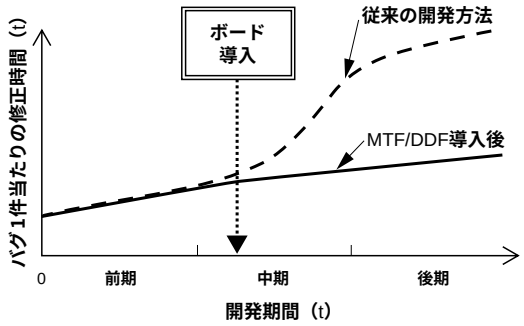

図9．各開発工程における不具合修正時間　従来開発とは，開発後期においては実ボードだけを使用した場合を表しており，MTF/DDF導入開発では図7で示した開発環境を使用した場合を指している。
Defect correction time in each development phase

5 あとがき

組込みシステム開発においても，上流工程において十分な分析・設計を行い適切なアーキテクチャを構築することの重要性を確認できた。また，その結果を容易に展開する技術としてMTF/DDFの有効性を示すことができた。今後は，MTF/DDFの他MPUへの展開と協業による顧客との連携を通じて，組込みシステム開発効率改善に貢献していく。

謝　辞

協業及びこの論文への掲載にあたり，多大なるご協力をいただいた富士ゼロックス(株)ドキュメント プロダクトカンパニー 商品開発統括部 第二事業商品開発部の関係各位に感謝の意を表します。（宮田／玉木）

文　献

⑴ 玉木裕二，ほか．“組込みシステム向けオブジェクト指向フレームワークの開発”．第3回組込みシステム技術に関するサマーワークショップ（SWEST3），2001-07.
⑵ 坂村　健監修，μITRON 3.0標準ガイドブック改定新版，パーソナルメディア，1997，427p.


深谷　哲司　FUKAYA Tetsuji
研究開発センター　システム技術ラボラトリー研究主務。
ソフトウェア生産技術の研究・開発に従事。情報処理学会会員。
System Engineering Lab.


吉田　和樹　YOSHIDA Kazuki
e-ソリューション社 SI技術開発センター SI技術担当主務。
C Solutionのアプリケーションフレームワークの開発に従事。情報処理学会，日本ソフトウェア科学会会員。
Systems Integration Technology Center


細谷　竜一　HOSOYA Ryuichi
e-ソリューション社 SI技術開発センター SI技術担当。
C Solutionのアプリケーションフレームワークの開発に従事。情報処理学会会員。
Systems Integration Technology Center


宮田　尚志　MIYATA Takashi
セミコンダクター社 マイクロプロセッサ統括部 マイクロプロセッサ ソフトウェア担当。
オブジェクト指向技術，組込みシステム開発環境の研究・開発に従事。情報処理学会会員。
Microprocessor Div.


玉木　裕二　TAMAKI Yuji
研究開発センター　システム技術ラボラトリー研究主務。
ソフトウェア工学の研究，主に組込みソフトウェアの開発コンサルタントに従事。情報処理学会会員。
System Engineering Lab.