オブジェクト指向による組込みシステム開発のご紹介

(株)オージス総研 オブジェクト第一事業部 開発技術コンサルティング室
渡辺 博之

■ はじめに

最近では、組込み機器の開発にもオブジェクト指向を使おう、という動きが盛んになってきています。オブジェクト指向自体は、ここ数年で一気にその認知度を増し、特にビジネス系システムや PC のアプリケーション開発などではかなりの有効性が証明され多くの実績も出来てきました。

組込みシステム開発に携わるエンジニアの方たちも、それらの状況を徐々に感じているようで、自分たちの開発にもオブジェクト指向のメリットを活かせないか、と少しずつ考え始めているようです。しかし、「ビジネス系」と「組込み系」ではシステム開発の特徴や対象が大きく異なります。ビジネス系でうまくいったことが組込み系で必ずしもうまく当てはまるとは限りません。そういった疑問から、オブジェクト指向開発の導入に二の足を踏んでいる方も多いかと思います。

今回は、そんなことを念頭に置きながら、「組込みシステム開発の抱える問題点」そして、それを解決する上で「オブジェクト指向は本当に有効な開発手法なのか」を中心に考えてみたいと思います。

■ 組込みシステムの抱える問題点

私自身が過去に関わった開発や、今までおつきあいしてきたお客様の話をまとめると、組込みシステム開発の抱える問題は以下のようになります。

◇ 組込みシステムは、ハードウェア制御や効率重視といった特徴を持つため、「とりあえず動くもの」が求められる傾向にある。そのため、システムは機能中心の「動く(動いた?)」モジュールのつなぎ合わせになりがちで、その結果、システム全体の構造(アーキテクチャ)が不在となり、非常に見通しの悪いシステムとなってしまう。変更への対応などが加わることにより、この傾向はよりいっそう加速される。

◇ 前述した理由により、システムが効率中心やアーキテクチャ不在

で作成されているため、変更や修正が難しい。結果的にこれらの作業が新規開発の工数を圧迫するため、また同じようなやり方でシステムを作ってしまうという悪循環が続いてしまう。

◇ システムの単位が機能中心でまとめられたタスクであるため再利用がしづらい。また、タスク自体の粒度が大きすぎる点も再利用性を阻む要因となっている。

◇ ビジネス系のシステムと比較すると、サイズの小さいものが多いため、きちんとした開発プロセスの必要性が薄く、実装中心で力まかせの開発形態になりやすい。設計のノウハウなども担当者の経験としてのみ残され、職人芸的な開発体制に依存する結果を生む。

また、最近は組込み機器自体に占めるソフトウェアの割合が高まっており、その結果として次のような問題も深刻化しています。

◇ 組込みシステム自体が大規模化・複雑化しているため、従来はあまり必要のなかったユーザーインタフェースやネットワーク機能などが必須になりつつある。また、取り扱うデータも大きく複雑になってきており、従来のイベントドリブンのハードウェア制御といった観点からの設計だけでは対応できなくなりつつある。

これらの問題を解決するためには、

☆ システム全体の構造（アーキテクチャ）をきちんと設計できること

☆ 拡張性や変更に強いこと

☆ 動作を確認しながら開発できること

☆ 大規模・複雑化に対応できること

といった要件を満たした開発形態に移行する必要があります。しかし、先に見てきたように、従来の職人芸スタイルや機能中心の開発手法ではこれらの要請に応えることができませんでした。そこで登場したのが、オブジェクト指向による開発です。果たしてオブジェクト指向開発はこれらの要件を満たしてくれるのでしょうか?

■ オブジェクト指向開発の特徴

まず始めに、オブジェクト指向開発とはどんなものかを簡単に紹介していきたいと思います。一般的な特徴を簡単にまとめると次のようになります。

● 機能中心ではなく現実世界の構

造や概念を中心に据える
- アーキテクチャ中心の開発を行う
- インタフェースと実装を分離する
- 繰り返し型の開発を行う
- UML という統一されたモデリング技法を使用する

以降では、これらのアプローチひとつひとつの紹介とそれらから得られるメリット、特に組込みシステムに対する有効性について具体的に考えて行きたいと思います。

■ 現実世界の構造や概念を中心に据える

よく言われるように、オブジェクト指向では、現実の世界を忠実に再現することが基本となります。例えば、通信システムに必須の電文はそのまま電文オブジェクトとして切り出されます。電文オブジェクトは電文に関する機能を持っているので、今までであれば第 3 者のモジュールがゴリゴリ行っていた電文解析のような処理も、電文オブジェクトに「解析しろ」とメッセージを送るだけで済んでしまいます。ですから電文のフォーマットが変わっても、電文オブジェクトだけがそれを知っていればいいことになり、結果的に変更に強いシステムになります。

また、オブジェクトの基本は機能ではなく既存のものや概念ですから、オブジェクト自体は機能に影響されない非常に安定したものとなります。先の例で言えば電文オブジェクトのもつ電文解析といった機能はシステムの機能に関係ない汎用的なものです。システムはこれら汎用的で安定したオブジェクトの機能をいかに組み合わせるか、で実現されることになります。組込みシステムでよく使用されるタスクも複数のオブジェクトから構成されるのが一般的です。

■ アーキテクチャ中心の開発

先に述べたように、オブジェクト指向により開発されたシステムは、基本的なオブジェクトの組み合わせから成り立ちます。しかし、これは組込みシステムの問題点で述べたように、つなぎ合わせのシステムになりやすく、その結果としてシステム全体の構造が見えにくい、という問題点を持っています。最近のオブジェクト指向開発では、これらの問題点に対処するために、アーキテクチャを中心にした開発形態をとっています。

アーキテクチャとは、システム全体

の構造や制御方法、共通してつかわれるメカニズムなどを指します。具体的には、システムの構造をオブジェクトの性質毎にグループ化し階層化したり、システム全体のタスク構成を決めたり、イベントの送受信メカニズムを決めたり、といった内容を指します。これらのアーキテクチャを明確に定義することで、秩序だったわかりやすいシステム構成が得られるようになります。表記法としても、パッケージやサブシステムといったオブジェクトの集合となる概念が用意されています。

■ インタフェースと実装の分離

よくオブジェクト指向は再利用性があると言われますが、その理由の一つに、インタフェースと実装の分離が挙げられます。呼び出し側で特定のオブジェクトを指定せずインタフェースだけを指定して呼び出しを行うことで、呼び出されるオブジェクトを実行時に決定することができます。つまり、インタフェースを介することで、呼び出し側と呼び出し先のオブジェクト間の依存関係を切り離すことができます。例えば、ハードウェア制御を行うオブジェクトなどをインタフェースと実装を分離した設計にすることで、ハードウェアの機種変更や仕様変更などからシステムが受ける影響を減らすことができるようになります。

■ 繰り返し型の開発

オブジェクト指向では構造化手法などとは異なり、分析から設計まで同じモデルやドキュメントを使用できます。この特徴を最大限に活かしたのがオブジェクト指向でよく使用される繰り返し型の開発です。これは、すべての要件を一度に開発するのではなく、分析から実装までを 1 サイクルとし、リスクに応じ複数のサイクルに分けてシステムを開発しようというやり方です。

組込みシステムの開発では、新しいハードウェアの特性を調べたり実際の効率を検証する上でも、繰り返し型により実際に物を動かしながら開発を進めていくやり方が、非常にマッチします。

■ UMLという統一モデリング技法

オブジェクト指向開発では UML という統一のモデリング技法を使用します。これにより今までであれば表現がまちまちで共有しにくかった、並行性や分散性、時間制限などといった組

込みシステム特有の問題に対する設計意図を、実装時ではなく設計時点で共有する事が可能になります。

■ オブジェクト指向の有効性

いままでの話を元に、ここでは、組込みシステムの問題点をオブジェクト指向開発で解決できるのか、といった点について考えてみたいと思います。

先に、組込みシステム開発への課題として以下のような項目を挙げました。

- システム全体の構造(アーキテクチャ)をきちんと設計できること
- 拡張性や変更に強いこと
- 動作を確認しながら開発できること
- 大規模・複雑化に対応できること

最後の点を除いては、オブジェクト指向開発の特徴で紹介した内容で大体の有効性は分かっていただけたかと思いますので、ここでは、最後の点に絞って考えてみたいと思います。

大規模化・複雑化への傾向は、特にここ数年で激しくなっていますが、このことが引き起こす問題として 2 点あげることができます。1 点目は、GUI やネットワークなど今まで知らなかった問題領域への対応です。今までの組込みシステムは特定の問題領域を一貫して扱うことが多く、分析作業はあまり必要とされませんでした。ところが、新たな問題領域への対応するためには正しい分析作業が必須となります。構造化手法などとは異なり、オブジェクト指向での分析は問題領域の本質に基づいたやり方を取るため、こういった新たな問題領域には非常に向いているといえます。

問題の 2 点目は組込みシステムが取り扱うデータ自体も大きく複雑になってきたことです。従来のようにデータと処理が分離していると、データ構造の複雑さや変更がシステム内の多くの部分に影響を与えてしまいます。これらのデータをオブジェクトとしてしまうことで、構造の複雑さや変更からくる影響をオブジェクト内だけに隠蔽することが可能になります。

■ オブジェクト指向開発のイメージ

「さて、現状の開発の問題点も分かったし、オブジェクト指向のメリットも分かった。だがいったい何から始めればいいんだろう?」

ここでは、そういった方々へ具体的なオブジェクト指向開発のイメージを

ご紹介したいと思います。

オブジェクト指向開発は次のようなフェーズに分かれて進んでいきます。

- 要求分析
  システムに求められる要求をまとめる。ユースケースといった機能の表現方法や、特に組込み系では従来から使用されているイベント解析なども使用される。
- 分析
  システムが扱う対象領域をオブジェクトとして表現する。これにより、問題領域の理解を深めることができると同時に、ここから得られたオブジェクトはそのまま以降の開発の中心となる。
- アーキテクチャ設計
  オブジェクトより大きい枠組みでのシステムの構造や制御方法、スレッドポリシー、イベント、排他制御メカニズムなどを決定する。
- システム設計
  サブシステム単位での設計とタスクへのマッピングなどを検討する。
- オブジェクト設計
  オブジェクト単位での設計とタスクへのマッピングなどを検討する。組込みシステムで旧来より使われてきた状態図や状態遷移表などのモデルはこの時点でオブジェクト単位に作成される。また、要求分析で導出した機能を実現するオブジェクト同士での協調動作を設計する。
- 実装
  各オブジェクトの元になるクラスの宣言と定義を実装する。

構造化手法との一番大きな違いとしては、基本的に各フェーズで作成されるモデルが同じであるため、フェーズ間での移動が容易なことが挙げられます。

なお、これらの作業は通常、オブジェクト指向開発をサポートする CASE ツールを使って行われます。CASE ツールを使うことのメリットとしては、まず UML でサポートされている様々なモデル（図式）の記述が簡単になることですが、それ以外にも、モデルで表されたオブジェクトの定義や、それらの関係などをリポジトリと呼ばれるデータベースのようなもので一括して管理してくれる点があります。これにより、設計者が 1 つのモデルを修正すると自動的に他のモデルにも変更が反映されることが可能になります。

また、モデルからの自動コード生成機能や、その逆のソースコードからの

モデル作成機能なども備えています。通常のオブジェクト指向開発ではオブジェクトの数が非常に多くなるため、モデルから手動でコードを書き起こすことはあまり現実的ではありません。CASE ツールが提供するコード生成機能を使用するのが一般的です。

■ RoseとZIPC

オージス総研では先に述べたようなオブジェクト指向開発のための CASE ツールとして Rational 社の Rational Rose という製品を扱っています。Rational 社はオブジェクト指向の標準モデリング技法である UML を生み出した会社であり、オブジェクト指向方法論者として有名なブーチ、ランボー、ヤコブソンらを抱えたトップクラスの開発ツールベンダーです。そういった背景もあり Rational Rose は世界中でオブジェクト指向開発用 CASE ツールとしてトップシェアを誇り、事実上の標準ツールとなっています。

Rose を使用することで、前述したような開発形態を非常に効率よく進めていくことができるようになりますが、実際に組込みシステム開発で使おうとした場合には、オブジェクトの状態モデル作成機能に多少の不満を感じる場合があります。まず、UML では状態図をサポートしていますが、状態やイベントが非常に多いものや例外処理の多いものなどでは記述や検証が困難になりがちです。また、現状の Rose では状態図で記述された情報は自動コード生成に反映されません。

そういった状況を受け、昨年秋頃から、Rose の状態モデルを取り込んで ZIPC 上で状態遷移表として設計してしまおうという計画が進められています。これが実現すれば現在の Rose で多少手薄な状態モデルの設計を詳細に行うことができるようになり、オブジェクト指向開発のメリットをさらに高めてくれるでしょう。

■ 最後に

以上、駆け足でオブジェクト指向開発の特徴とメリットを述べてきました。

最近はオージス総研でも組込みシステムに関するオブジェクト指向での開発案件が多くなってきています。組込みシステムに関するオブジェクト指向開発は、これから実践の局面を迎えようとしているようです。本稿が組込みエンジニアであるみなさんのオブジェクト指向への理解に少しでも役立てたなら幸いです。

（わたなべ ひろゆき）