

ZIPCを用いた家電組込みソフトウェア開発の効率化

九州日立マクセル株式会社
新分野開発PT ソフトアドバンスGr

安部田 章

1. はじめに

近年家電製品が高度化し、そこに組み込まれるソフトウェアが大規模複雑化してきている¹⁾。その一方で、市場の要求に即応することが求められており、開発の効率化と信頼性確保が急務となっている。そこで弊社では、再利用性や保守性に優れ、早期開発に適したコンポーネントベース開発環境の構築をすすめている³⁾。

コンポーネントベース開発では、ソフトウェアフレームワークを基盤として、その上で利用可能なコンポーネントを準備し、それらと呼び出すことでプログラムを作成する。そこで、当社では、自社製品における家電組込みソフトウェアのドメイン分析⁶⁾を行うことにより、自社ソフトウェアの移植性や再利用性を向上させる多層アーキテクチャフレームワークを導出し、フレームワーク上で動作するコンポーネント群を整備した。

組込みソフトウェアの移植性を考えたとき、まず、マイコン間の移植性と、ハードウェア間の移植性を向上させる必要がある。そこで、我々はソフトウェアの構成をマイコンハードウェアドライバ層、メカニズム層、アプリケーション層の3つのレイヤーに分割し、各階層の特性を考慮した部品化を行うことにより、効果的な再利用を目指した。

まず、マイコンハードウェアドライバ層は、マイコンのハードウェアに依存する部分を抽象化し、マイコン機種に依存しない共通なインターフェースを持つドライバを整備することでマイコン間の移植性を高めた。

メカニズム層は、制御対象のハードウェア毎にその制御メカニズムをコンポーネント化し、ハードウェア部品単位で再利用可能にした。また、ハードウェアの種別や制御構造に依存しない共通なインターフェースを提供することでハー

ドウェア間の移植性を高めた。さらに組込みソフトウェアではハードウェア制御コンポーネントが大規模複雑化する傾向がある⁴⁾ため、階層化コンポーネントとして構築することにより、メカニズム層の再利用性を高めた⁵⁾。

家電組込みソフトウェアの場合、アプリケーション層は、製品の振る舞いを実現するため、状態によって挙動が異なる状態遷移機構としてモデル化できる。そこで、アプリケーションをいくつかの状態遷移機構としてモデル化し、アプリケーションが提供する機能を実現するコンポーネント群を準備しておく構成とした。そして、状態遷移機構における状態アクティビティと状態遷移アクションから、機能コンポーネントのサービスと呼び出すことで実装を行う。この状態遷移機構のモデル化および保守管理には、ZIPCを用いることにより、仕様漏れがなく、仕様変更を効率よく行えるアプリケーション層の開発を実現した。

本稿では、特に、アプリケーション層の構成方法とツールを用いた構築・管理支援について説明し、多層アーキテクチャフレームワークとツール利用により開発効率を向上させた事例について紹介する。

2. 多層アーキテクチャフレームワーク

本フレームワークでは、図1に示すように、基本アーキテクチャを共通化し、マイコンハードウェアドライバ層、メカニズム層、アプリケーション層を別レイヤーとして構成する。マイコンのハードウェアに依存する部分()と、機構および回路の制御メカニズムに依存している部分()を別レイヤとすることにより、マイコン機種変更時は、マイコンハードウェアドライバ部分のみを交換すればよい上に、機構や回路部品の変更時にも、マイコンハードウ

ェア部分の修正がなく、メカニズム層の制御対象ハードウェアに対応する部分のみを交換すればよい。



図1：フレームワーク構成図

これにより、マイコン機種変更、メカや回路のハードウェア変更時による影響の範囲を局在化させ、移植作業の効率化と信頼性の向上を実現することができた。また、アプリケーション層(#)が分離されているため、これらの変更時には、アプリケーション層の変更が基本的には生じないようにした。

2.1. メカニズム層のコンポーネント化

組込み制御ソフトウェアのメカニズム層をコンポーネント指向で作成する場合、制御対象のハードウェアに一つの制御オブジェクトを割り当て、その制御オブジェクトがいくつかのオブジェクトを統制して一つのまとまりのある制御メカニズムを実現させる。それらの制御オブジェクトを中心としたオブジェクト群を一塊のコンポーネントとして作成し、コンポーネント単位で扱えるようにする。

これにより、ハードウェア部品の交換時に、その影響範囲をコンポーネント内に局在化できるため、メカニズム層の移植性が向上した。また、開発後には、これらのコンポーネントをフレームワークに蓄積していくことにより、次の製品開発ではコンポーネント単位で容易に再利用が可能になった。

2.2. アプリケーション層の再利用

一般に業務系ソフトウェアのアプリケーション層は、図2に示すようにシステムの提供する機能単位に処理を分割することができる。その場

合、独立性の高い機能単位で追加・変更・削除を行うことが可能である。

これに対して、制御系ソフトウェアのアプリケーション層は、システムの振る舞いを実現する部分であるため、システムの動作モード単位に処理を分割することになる。



図2：機能分割によるプログラム構成

すなわち、制御系ソフトウェアの場合、システムの振る舞いが動作モードによって変化する特性を持ち、いくつかの動作モードとその遷移構造がシステムの機能を実現している。したがって、動作モードの追加・変更・削除と、そのモード遷移構造の変更を容易に行える必要がある。

そこで当社では、図3に示すように、アプリケーション層の上位層を、状態遷移機構(動作モード、イベント、遷移およびアクション)として定義し、動作モードごとに機能分割する状態遷移フレームワークとして構成する。アプリケーション層の下層には、動作モードを実現するのに必要な機能をコンポーネント化して準備し、これらを状態遷移フレームワークから呼び出すことによってアプリケーション層を構築する。

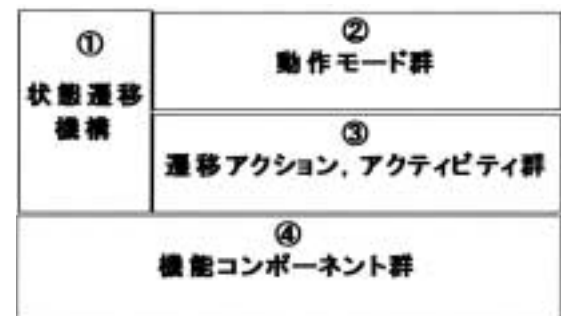


図3：アプリケーション層の構成

本構成にすることにより、製品の振る舞いの部分に変更があった場合、動作モードの追加・変更・削除によって容易に行えるようになる。さらに、提供する機能に変更があった場合には、機能コンポーネントのみを修正すればよく、状態遷移構造には影響を与えず、仕様変更に強いプログラムを作成できるようになった。

3. ZIPCによるアプリケーション層の構築

アプリケーション層の上位層の構造は、図3に示すように、状態遷移機構を実現する部分、動作モード群 状態遷移機構から呼び出されるアクションやアクティビティから構成される。アクションは、イベント発生や状態遷移時に実行されるべき機能や処理であり、アクティビティは、特定の状態がアクティブな間中、繰り返し実行される機能や処理である⁷⁾。

下位層は、これらのアクションやアクティビティから呼び出される機能コンポーネント群()から構成される。機能コンポーネントは、モード非依存なコンポーネントとして作成する。遷移アクションやアクティビティからは、機能コンポーネントのインターフェース関数を呼び出すことで、製品の振る舞いを実現する。

上位層の状態遷移フレームワークの部分を下位層の機能コンポーネント群から分離することにより、動作モードの変更の影響を機能コンポーネントから排除した。これにより、機能コンポーネントの独立性を高め、アプリケーション層の機能単位での再利用性を高めることができた。

さらに、製品の仕様変更による状態遷移機械への動作モードの追加、削除および変更などの操作・管理を専用ツールで行うようにすることで、製品機能の追加、削除および変更作業を大きく効率化することができた。

本状態遷移フレームワークを実装するために、当社ではZIPCを利用している。ZIPCは、状態遷移表を作成することにより、動作モードとその遷移構造を容易に実装することができる。さらに、遷移アクション、モードアクティビティなどフレームワークで必要なすべての関数をツール上で定義することができる。作成した状態遷移表からC言語のソースを自動生成することができる。自動生成されたソースは実際のアプリケーションの状態遷移フレームワークとして

そのまま利用することができる。

状態遷移フレームワークをZIPCを用いて実装することによる利点としてさらに、仕様変更が状態遷移表の変更といったモデルベースで行えるようになり、変更に必要な工数や変更ミスを削減できることがあげられる。

以上により、製品のラインアップやモデルチェンジ時は、モデル間の動作モードの追加、削除および変更作業をZIPC上で行うことにより効率化することができるとともに、アプリケーション層の機能単位の再利用性を高めることができた。また、マイナーチェンジの場合、ベースモデルのアプリケーション層全体を再利用し、差分となる動作モードをZIPC上で追加、削除、変更することにより、最小限のコストで開発することができるようになった。

4. 本フレームワークの全体評価

本構成により、実際に健康機器を構築した場合の開発効率を示す。

図4に健康機器の旧型から新型へのモデルチェンジを行った場合の動作モードの追加・変更・削除を一覧にしたものを示した。モデルチェンジ品の開発は、ZIPC上で複数の動作モードを追加変更し、遷移先の変更、アクションやアクティビティの変更を行い、C言語のソースコードを生成することによって行った。これにより、多くの部分を再利用できるとともに、変更もZIPC上で簡単に行うことができた。



図4：動作モードの追加・変更

変更手順としては、以下の流れで行った。

- 動作モードの追加・変更
- モード遷移アクションおよびアクティビティの追加・変更
- 機能コンポーネントの追加・変更
- 状態遷移フレームワークとしてZIPCからCコード再生成
- プログラムのアプリケーション層に状態遷移フレームワークを組み込む

ZIPCを利用して本フレームワークを適用した場合の開発期間の評価をおこなった。表1に本フレームワークの適用/非適用時の開発期間を比較した。旧型から新型へのモデルチェンジにおいて、適用/非適用を比較したものだが、適用事例と非適用事例で機種が異なるため、厳密な評価はできないが、フレームワークを適用することによって開発期間が少なからず短縮されたことが示唆された。また、ZIPC導入による工数削減の評価であるが、これに関しては直接評価できるデータがないことが残念であるが、フレームワーク非適用事例では、ZIPCを導入しておらず、適用事例ではフレームワーク適用とともに導入している。ZIPCを導入しなければ、フレームワークの作成と管理が煩雑になるためである。

表1：開発期間の評価

	ステップ数:A	開発期間 :B	A / B
フレームワーク非適用	約1.6万	約150日	10.7
フレームワーク適用	約2万	約75日	2.67

5. 今後の課題

今後はキャッツ株式会社製のDrawrialを利用してリモコンLCDの画面設計と実装の簡易化を進めることにしている。近年世界的な健康ブームから当社健康関連の開発製品を海外で販売する要求が高まっており、LCD表示の多言語化は必須となってきている。Drawrialは多言語開発に対応していることもあり、導入を決定した。また、現在、制御コンポーネントのモデルベースの開発を試行している。状態遷移構造を持つ制御コンポーネントを、ZIPCあるいはKones-RealTimeによる状態遷移機械としてモデル化し、モデルベースでの再利用を行おうというものである。以上の取り組みに関しては、次回以降に報告をしたいと考えている。

参考文献

- 1) 高田広章：組込みシステム開発技術の現状と展望、情報処理論文誌、Vol.42, No.4, pp.930-938 (2001)
- 2) 渡辺政彦：状態遷移ベースのソフトウェア開発環境の現状と動向、計測と制御、Vol.41, No.2, pp.117-121 (2002)
- 3) 安部田章：組込みソフトウェアの移植性を向上する多層アーキテクチャ、ウィンターワークショップ神戸論文集、情報処理学会シンポジウムシリーズ、Vol.2003, No.5, pp.9-10 (2003)
- 4) 渡辺博之、渡辺政彦他2名：組み込みUML、翔泳社、pp.129-132.
- 5) 安部田章：多層化による組込みソフトウェアの移植性の向上、ソフトウェア工学研究会資料、02-SE-140-16, pp.117-122 (2003)
- 6) 安部田章、時任正博、島田和明：組込みソフトウェアのドメイン分析に基づく多層アーキテクチャフレームワーク、組込みソフトウェアシンポジウム2003 (ESS2003) 論文集、情報処理学会シンポジウムシリーズ、Vol.2003, No.13, pp.14-21.
- 7) 渡辺政彦：拡張階層化状態遷移表設計手法 Ver.2.0, 東銀座出版