

携帯電話制御ソフトウェアのアスペクト指向実現

坂野将秀[†] 久松康倫[†] 本多克典[†]
水野耕太[†] 野呂昌満[‡]

[†] 南山大学大学院数理情報研究科数理情報専攻

〒 489-0863 愛知県瀬戸市せいれい町 27

[‡] 南山大学数理情報学部情報通信学科

〒 489-0863 愛知県瀬戸市せいれい町 27

本研究では、携帯電話制御ソフトウェアの事例において、どのような横断的コンサーンが存在するかを考察し、アスペクト指向開発を適用する。横断的コンサーンに関する処理を局所化することによりソフトウェアの柔軟性が向上することを確認する。携帯電話制御ソフトウェアは典型的な組み込みソフトウェアである。この事例におけるソフトウェアの構造を一般化し、他の組み込みソフトウェアに適用し考察する。

An Aspect-Oriented Implementation of a Cellular Phone Control System

Masahide Banno[†], Yasunori Hisamatsu[†], Katsunori Honda[†],
Kota Mizuno[†], and Masami Noro[‡]

[†] Master's Program in Mathematical Sciences and Information Engineering, Nanzan University graduate school
27 Seirei-cho, Seto, Aichi, 489-0863, Japan

[‡] Dept. of Information and Telecommunication Engineering, Nanzan University
27 Seirei-cho, Seto, Aichi, 489-0863, Japan

The paper gives what kind of concern can be detected during the design of Cellular Phone Control System. It is discussed that the Aspect-Oriented desing based on the concerns promotes the flexibility of the system. We tried to generalize the software and discuss the general structure which Embedded systems have to have.

1 はじめに

横断的コンサーンを分離、記述するためのアスペクト指向に関する研究が盛んに行われている。アスペクト指向開発では、横断的コンサーンを分離することにより、柔軟性、再利用性の高いソフトウェアの開発が可能だといわれている。

本研究では、携帯電話制御ソフトウェアの事例において、どのような横断的コンサーンが存在するかを考察し、アスペクト指向開発を適用する。まず、オブジェクト指向開発を行い、横断的コンサーンを抽出する。つぎに、これらのコンサーンを基にアスペクトに分割を行う。その後、オブジェクト指向開発したソフトウェアと比較し、柔軟性の向上を確認する。

携帯電話制御ソフトウェアは典型的な組み込みソフトウェアである。この事例における横断的コンサーンの多くは他の組み込みソフトウェアにも当てはまると考えている。本研究の成果を一般化することで、組み込みソフトウェアのアスペクト指向アーキテクチャが得られるはずである。

2 携帯電話制御ソフトウェアのオブジェクト指向開発

携帯電話制御ソフトウェアにおける横断的コンサーンを確認するためにオブジェクト指向開発を行った。

2.1 分析

現在使用している携帯電話とその取扱説明書をもとに、分析を行った。携帯電話は対話型システムであること、複数のサブシステムによって構成されていることを確認した。

対話型システム

携帯電話制御ソフトウェアは対話型システムである。入力として、ボタンからの入力、ネットワークからの入力と考えられる。入力がおこると、携帯電話はイベントを受け取り、出力を行う。出力として、画面への出力、ネットワークへの出力と考えられる。

サブシステム

携帯電話は複数のサブシステムにより構成される。分析した携帯電話に共通していたサブシステムを以下に示す。

- 通話システム
- メールシステム
- Web 閲覧システム
- ファイルシステム
- アドレス帳システム
- カメラシステム

- テキスト編集システム
- 設定システム

2.2 設計と実現

分析した結果をもとに、携帯電話制御ソフトウェアをオブジェクト指向設計、実現する。携帯電話制御ソフトウェアは対話型システムなので、MVC モデル [3] を適用することにより構造を整理できる。また、サブシステムの追加や管理を容易にするために、各サブシステムは多相性を用いた共通な構造とした。実現には、オブジェクト指向言語として広く使用されている Java[4] を利用した。

MVC モデル

MVC モデルの適用により、アプリケーション実行部、出力、入力をモデル、ビュー、コントローラとして分割できる。これによりそれぞれの独立性が向上する。MVC モデルの実現には、Observer パターン [2] を利用している。また、コントローラの入力処理には Java のイベントモデルを利用している。

携帯電話制御ソフトウェアのクラス構造を図 1 に示す。CellularPhone はモデルの中核を形成し、サブシステムの管理を行う。OutputDevice はビューに相当し、出力処理を行う。InputDevice はコントローラに相当し、入力処理を行う。入力は同時に起こる可能性があるため、イベント管理のための疑似並行実行モニタ (Monitor) を用意している。

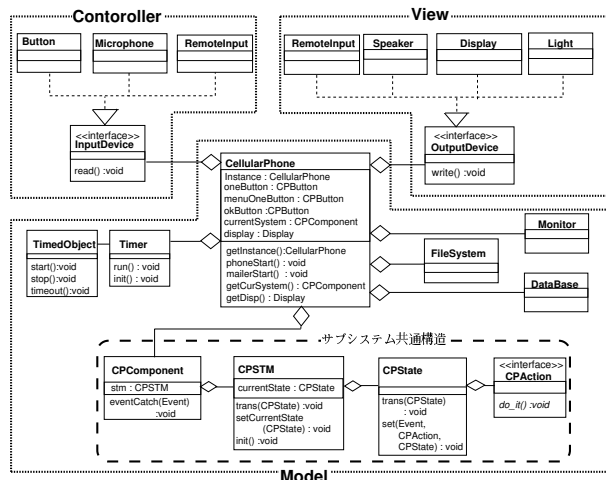


図 1: 携帯電話制御ソフトウェアのクラス図

サブシステム

サブシステムを多相性を用いた構造にすることで、サブシステムの動的な切替えや、追加が容易になる。各サブシステムは、状態によって異なる振舞いをする状態遷移機械である。状態の管理を

ンを分離するひとつの手段と考えている。MVC モデルのコントローラをイベント処理、ビューを外部出力コンサーンとして抽出した。

並行処理コンサーン

並行処理は、イベントを発生させるオブジェクトに横断的に関連している。本研究では、疑似並行実行モニタを用いて、並行処理を実現している。疑似並行実行モニタは、複数のイベントが同時に発生したとき、イベントを順に処理する。横断的に関連している様子を図 3 に示す。

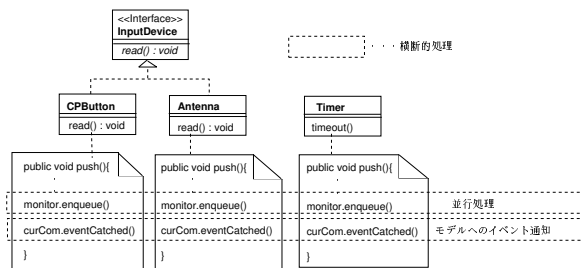


図 3: 並行処理に関する処理

メモリストレージコンサーン

メモリを制限する処理は、アドレス帳に登録する処理、撮影した画像を登録する処理などに横断的に関連している。携帯電話は記憶領域が限られているので、メモリの使用を制限する必要がある。カメラシステムをオブジェクト指向により実現した構造では、メモリを制限する処理が散在してしまう。横断的に関連している様子を図 4 に示す。

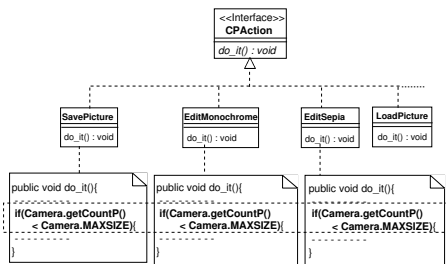


図 4: メモリストレージに関する処理

例外処理コンサーン

携帯電話における例外処理は、複数のオブジェクトに横断的に関連している。我々は今回、例外処理として、メモリ制限にともなう例外処理、通信の制御にともなう例外処理などを考えた。メモリ制限にともなう例外処理とは、データの新規登録時に登録上限数に達している場合、新規保存させない処理のことである。通信の制御にともなう

例外処理は、通信時に一定時間応答がない場合の通信の自動切断の例外処理などをいう。

実時間処理コンサーン

実時間処理は、時間計測が必要な複数のオブジェクトに横断的に関連している。ネットワークを介した通信を行うとき通信障害の検知のために、応答時間を計測する必要がある。また、通話するさいに通話時間を計測する必要がある。横断的に関連している様子を図 5 に示す。

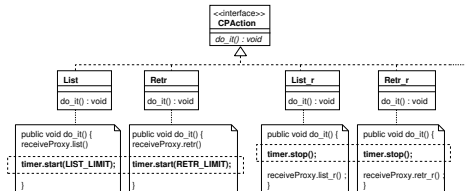


図 5: 実時間処理に関する処理

セキュリティ処理コンサーン

セキュリティ処理は、暗号化や復号化を行うオブジェクトに横断的に関連する。セキュリティ処理には、SSL などの通信プロトコルレベルと APOP などのアプリケーションレベルでの、暗号化、復号化がある。このようにセキュリティ処理は、散在している。

データ永続性コンサーン

データに永続性を持たせる処理は、データを扱う複数のオブジェクトに横断的に関連している。携帯電話が扱うデータとしては、アドレス帳の個人情報データ、メールシステムのメッセージデータ、カメラシステムの画像データなどがある。これらのデータの新規登録、既存のデータの変更時などでデータに永続性をもたせる処理が必要となる。データ永続性が横断的に関連している様子を図 6 に示す。

インスタンス生成コンサーン

ファイルシステムなどのようにインスタンス生成をするさい、特殊な処理を必要とするものがある。この場合インスタンスをひとつしか生成されないように保証する必要がある。オブジェクト指向による実現では、Singleton パターン [2] でファイルシステムのインスタンス生成を制限している。したがって、インスタンス生成を制限するための特殊な処理を呼び出す必要がある。ファイルシステムのインスタンス生成に関する処理が横断的に関連している様子を図 6 に示す。

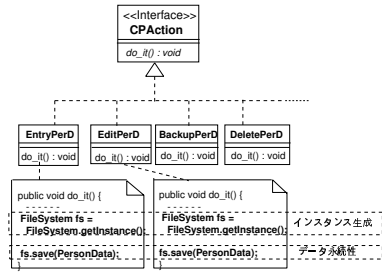


図 6: データ永続性とインスタンス生成に関する処理

3.2 横断的コンサーンにしたがった分割

抽出した横断的コンサーンにしたがってアスペクトに分割する。オブジェクト指向で実現したメールシステムをアスペクトに分割した様子を図 7 に示す。オブジェクト指向開発したメールシステムではセキュリティ処理は PDU クラス（図 2 参照）で行っている。Security は PDU からセキュリティ処理を分割して、モジュール化したものである。Exception は、例外に関する処理を、モジュール化したものである。

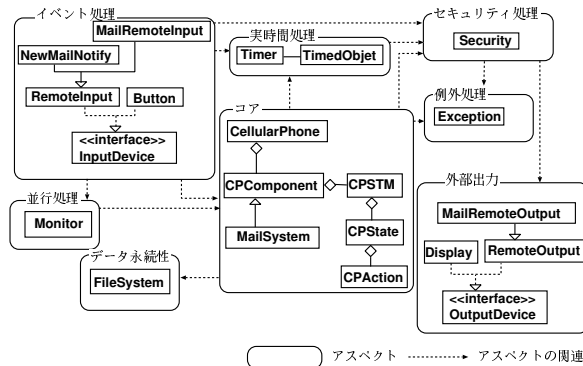


図 7: メールシステムのコンサーンにしたがった分割

アスペクトの階層

分割した複数のアスペクトにおいては、さらに以下の横断的コンサーンを確認できた。

- 状態遷移コンサーン
- アプリケーションロジックコンサーン

これを自然に記述するためには、アスペクトの階層化が必要である。イベント処理アスペクトでは状態遷移に関する処理と、アプリケーションロジックに関する処理が横断的に関連している。携帯電話のボタンは、押した時間の長さによって短押しと長押しというふたつの押し方がある。

ボタンが生成するイベントは短押しと長押しの場合で異なるので、ボタンは状態遷移機械となる。つまり、イベント処理アスペクトは、状態遷移アスペクトとアプリケーションロジックアスペクトに分割できる。並行処理アスペクト、例外処理アスペクトなども、イベント処理アスペクトと同様に分割できる。各アスペクトを状態遷移アスペクトとアプリケーションロジックアスペクトに分割した様子を図 8 に示す。

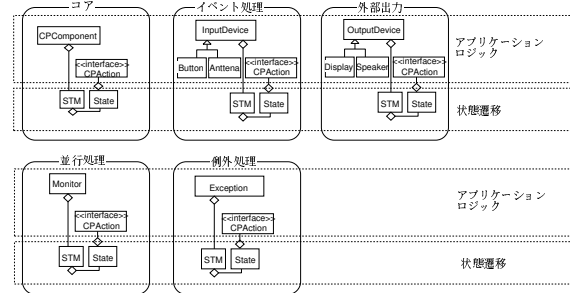


図 8: アプリケーションロジックと状態遷移の分割

協調場の適用

ロールモデル [5] では、ある文脈におけるオブジェクト間の関係をロール間の関係として、協調場に記述できる。携帯電話制御ソフトウェアでは、実現するサブシステムごとにオブジェクト間の関係が異なる。我々はサブシステムごとのオブジェクト間の関係を協調場で表現する。

図 9 はメールシステムと通話システムの協調場を表わしている。携帯電話を構成するオブジェクトは、協調場のロールと結合することで、他のオブジェクトとの関係が決まる。この場合、協調場はアスペクト間記述を表わしている。

3.3 実現

携帯電話制御ソフトウェアをアスペクト指向で実現する。アスペクト指向プログラミング言語のひとつで、Java の拡張言語である AspectJ[1] を使って実現する。実現した携帯電話制御ソフトウェアのシミュレータの実行の様子を図 10 に示す。

4 考察

オブジェクト指向とアスペクト指向で実現した携帯電話制御ソフトウェアを比較し、ソフトウェアの変更や追加に対する柔軟性について考察する。また、アスペクト指向開発した携帯電話制御ソフトウェアの構造を一般化し、他の組み込みソフトウェアへの適用可能性を考察する。

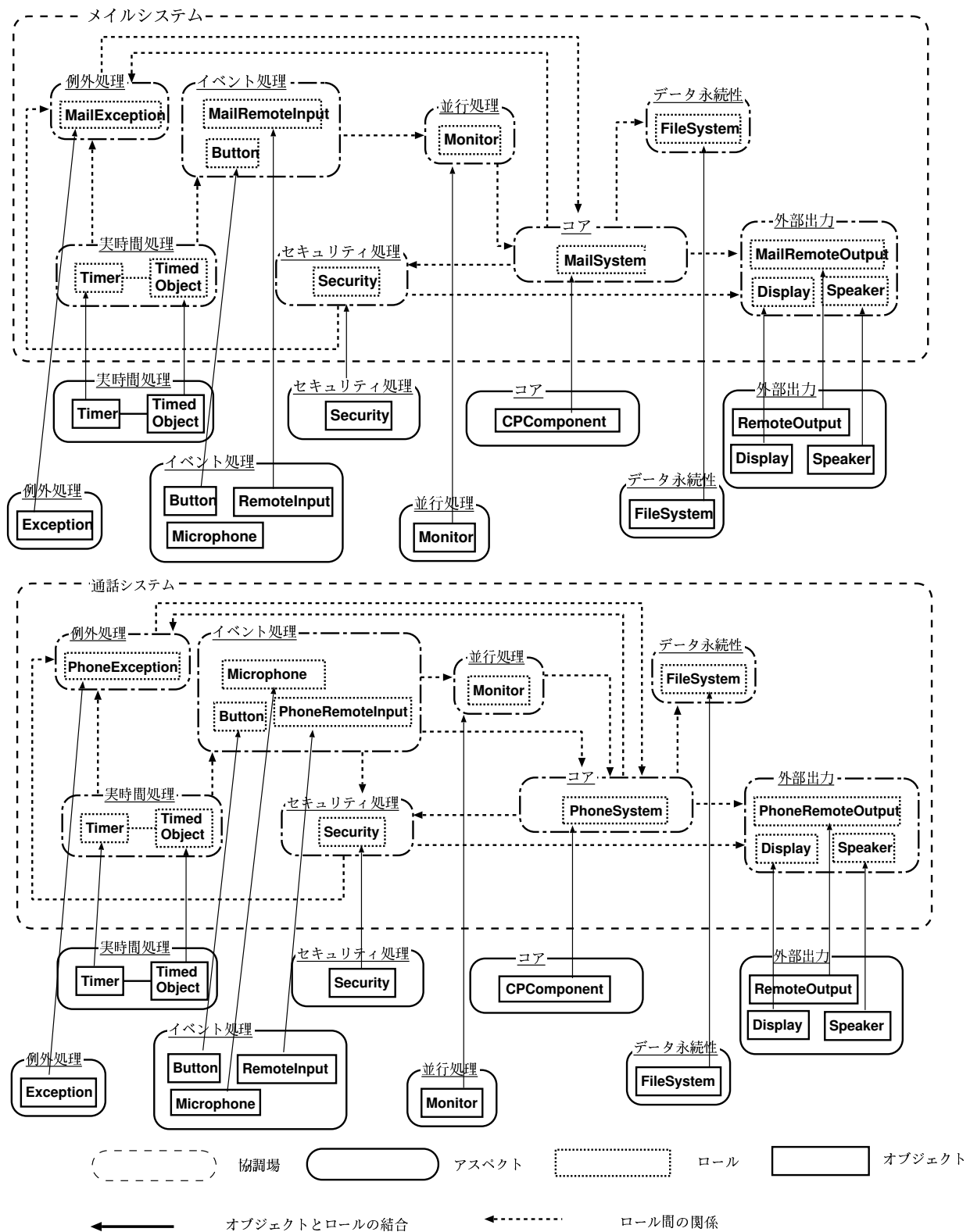


図 9: メールシステムと通話システムの協調場



図 10: 実現した携帯電話制御ソフトウェアの実行例

4.1 コンサーンごとの実現に関する考察

アスペクト指向実現した携帯電話制御ソフトウェアを基に、コンサーンごとの実現に関して考察を行う。

MVC モデルの実現に関する考察

オブジェクト指向で実現した携帯電話制御ソフトウェアは MVC モデルを適用した構造である。本研究では、コントローラの実現に Java のイベントモデルを利用し、InputDevice 自身をイベントリスナとした。イベントモデルでは、イベントリスナはイベントの通知先としてモデルを知る必要がある。入力装置の変更の例として、ボタンをタッチパネルに変更した場合を考える。この場合、タッチパネルの記述内に再度、モデルへのイベント通知を記述する必要がある。オブジェクト指向開発した携帯電話制御ソフトウェアにおいて、入力装置を変更したときの様子を図 11 に示す。

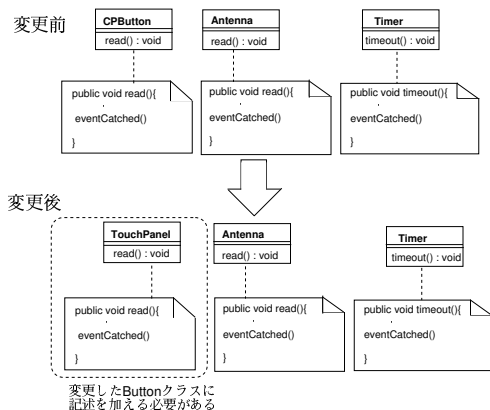


図 11: オブジェクト指向開発によるコントローラの変更

アスペクト指向実現では、コントローラとモデル間の関係をアスペクト間記述とし、局所化した。ボタンをタッチパネルに変更した場合を考える。変更点はアスペクト間記述に局所化されているので、変更したタッチパネルには記述する必要がある。アスペクト指向開発した携帯電話制御ソフトウェアのボタンをタッチパネルを変更した様子を図 12 に示す。

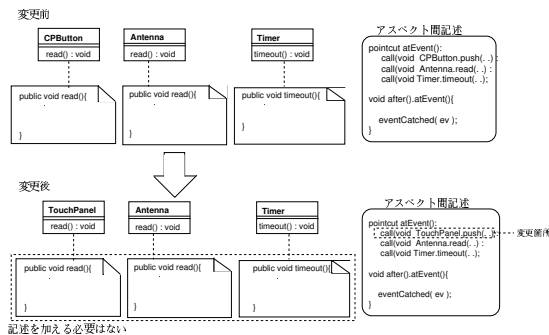


図 12: アスペクト指向開発によるコントロールの変更

モデルとビューの関係も、同様の局所化が可能である。これによりモデル、ビューとコントローラの独立性は向上する。

例外処理の実現に関する考察

本研究では、例外処理を Java の例外処理機構を用いないで実現した。組み込みソフトウェアでは例外が発生したさいに、行う処理が明確に決まっていなと、致命的な欠陥となる可能性がある。try, catch で記述した例外処理では、例外が発生した箇所や例外に対する処理などが特定できない。組み込みソフトウェアでの例外処理は動的な実現方法ではなく、静的な記述方法で実現すべきと考えた。例外処理を Java の例外処理機構を用いず記述し、アスペクト指向により分離し、局所化する。

実時間処理の実現に関する考察

オブジェクト指向開発では時間計測に関する記述が複数のクラスに散在している。携帯電話における通信ソフトウェアはサーバからの応答時間の計測を各コマンドに応じて行う。通信するさいのプロトコルの変更にともないコマンドの追加、変更を考えられる。オブジェクト指向で実現したメールシステムの各コマンドの送信を例にとり、実時間処理を変更した様子を図 13 に示す。

アスペクト指向では時間計測に関する記述は実時間処理アスペクト内に局所化した。したがって、プロトコル処理中に時間計測に関する記述をしな

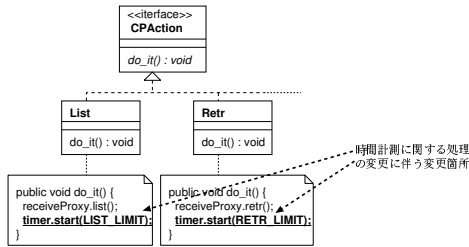


図 13: オブジェクト指向による実時間処理の変更

くてもよい。アスペクト指向による実時間処理の変更の様子を図 14 に示す。プロトコル処理を行うオブジェクト内に時間計測に関する記述がなくなったので、変更に対する柔軟性が向上したと考えられる。

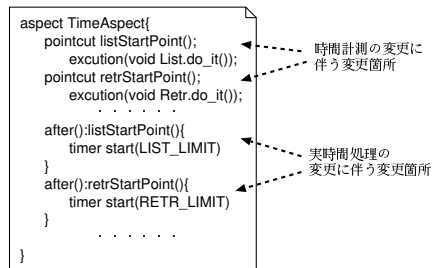


図 14: アスペクト指向による実時間処理の変更

4.2 組み込みソフトウェアへの適用

携帯電話制御ソフトウェアは組み込みソフトウェアの典型である。携帯電話制御ソフトウェアの事例において抽出した横断的コンサーンは他の組み込みソフトウェアにも当てはまると考えられる。例えば、自動販売機では、販売時には一定時間内に商品を出す必要があるため、実時間処理コンサーンが当てはまる。また、商品購入時に商品を出す処理、つり金を計算し、出す処理などは同時に行うために、複数のハードウェアを同時に制御する必要となるので並行処理コンサーンが必要となる。

以上から、本研究で実現した携帯電話制御ソフトウェアの構造は他の組み込みソフトウェアにも応用できるはずである。例えば、自動販売機制御ソフトウェアにおいて、実時間処理などの横断的コンサーンを分離すると図 15 のような構造が得られる。これにより、コンサーン毎の変更に柔軟な構造を得ることができる。

以上の議論を整理すると、図 15 の太線で示す組み込みソフトウェアの典型的な構造を得ることができる。アスペクトは容易に取り外すことができるので、各組み込みソフトウェアに応じてカスタマイズを行えばよい。これは組み込みソフトウェア

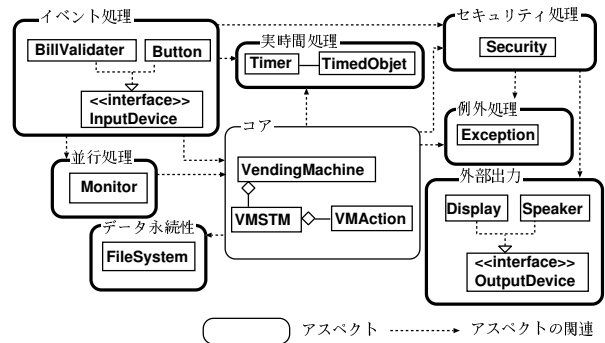


図 15: 自動販売機のコンサーンにしたがった分割

アに共通なアーキテクチャだといえる。

5 おわりに

本研究では、携帯電話制御ソフトウェアの開発にアスペクト指向が適用できることを確認した。まず、オブジェクト指向で実現した携帯電話制御ソフトウェアを基に横断的コンサーンを抽出した。その後、抽出した横断的コンサーンを基に、アスペクト指向を適用したことで、変更に対して柔軟性が向上したことを確認した。さらに、本研究で実現した携帯電話制御ソフトウェアの構造を一般化することで、組み込みソフトウェアのアーキテクチャを構築できると考えた。

参考文献

- [1] AspectJ, <http://eclipse.org/aspectj/>
- [2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*, Addison-Wesley,(1995).
- [3] Glenn E. Krasner, Stephen T. Pope, “A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80”, *Journal of Object-Oriented Programming*, Vol 1, Issue 3, pp.26-49,(1988).
- [4] Java, <http://www.java.com/>
- [5] T. Tamai: “Evolvable Programming based on Collaboration-Field and Role Model”, *International Workshop on Principles of Software Evolution*,(2002).