組み込みソフトウェアの階層化されたアスペクト指向ソフトウェアアーキテクチャ群の構築

Hierarchical Aspect-Oriented Software Architectures for Embedded Systems

熊崎 敦司* 野呂 昌満† 張 漢明‡

Summary.
Aspect-oriented technolgies, role based computation or field based computation have been developed for solving each problems. We integrated these technologies to aspect-oriented software architecture style. We tried to apply this architecture style to embedded systems to construct software architecture for embedded systems. As a result, we construct aspect-oriented software architecture to separate concerns without crosscut each other and active objects' behaviors.

1 はじめに

10 数年にわたる自動販売機制御ソフトウェアのオブジェクト指向開発 [7] を通して，我々はモジュール分割の基準が複数存在し，ある基準にしたがった分割が他の基準にしたがった分割と矛盾するという問題を認識した．これは近年注目を浴びている POP 技術，とくにアスペクト指向技術で解決可能な問題だと我々は認識している．他方，自動販売機の金銭や商品などは自動販売機中を移動するオブジェクトであり，場所によって振る舞いが変わる．場所毎にそのオブジェクトのすべてのインタフェースが必要ではないし，場所によってはインタフェースが不足することもある．このようなオブジェクトは，ロールに基づく計算 [10] や場を用いた計算 [12] で自然にモデル化できるといわれている．

我々は，別の応用領域もこの例に当てはまることを発見し，これらの考えを整理，アスペクト指向ソフトウェアアーキテクチャスタイルとして提案した．我々は簡素であることは大きな利点があるとの認識を持ち，合流点にメッセージを付加するというアスペクト指向で与えられている枠組みだけでこれらの技術を整理した．

本研究では，このアスペクト指向ソフトウェアアーキテクチャスタイルを組み込みソフトウェアに適用し，組み込みソフトウェアのアスペクト指向ソフトウェアアーキテクチャを構築する．これにより，組み込みソフトウェアの生産性，信頼性の向上を目指す．組み込みソフトウェアの応用領域は十分に大きいものなので，単一のソフトウェアアーキテクチャでは抽象度が高く実用的ではない．本研究では実用性を重視し，抽象度を鍵に階層化・整理したソフトウェアアーキテクチャ群を定義する．

事例として，自動販売機の金銭管理システムについて議論する．自動販売機の金銭管理システムのアスペクト指向ソフトウェアアーキテクチャでは，

– 横断的に関連する並行処理，実時間処理を矛盾なく記述できた，

*Atsushi Kumazaki, 南山大学情報管理学科
†Masami Noro, 南山大学数理情報学部情報通信学科
‡Han-Myung Chang, 南山大学数理情報学部情報通信学科

– ソフトウェア中を移動する金銭をオブジェクトとして，場所の変化にともなう振る舞いの変更を場所毎に分離，記述できた．

2 アスペクト指向ソフトウェアアーキテクチャスタイル

自動販売機制御ソフトウェアを例にアスペクト指向ソフトウェアアーキテクチャスタイルについて説明する．自動販売機制御ソフトウェア中には金銭のように，場所を移動し，場所毎に異なる振る舞いをするオブジェクトが存在する．金銭は販売時には代金として，払い出し時にはつり銭となる．このようなオブジェクトの振る舞いの変化は状態遷移機械として記述している．しかし，自動販売機制御ソフトウェアは状態遷移機械の集まりであり，場所の変化が複数の状態遷移機械に影響し，状態の遷移が複雑になるという問題がある．また，このオブジェクトの粒度が大きくなりすぎる．一般には，このような問題の解決策として，Epsilon [10] の Context を用いた計算や Flage [12] の場を用いた計算が提案されている．

その一方で，自動販売機制御ソフトウェアには複数のコンサーンが横断的に関連しているという問題がある．自動販売機制御ソフトウェアは並行動作する複数のハードウェアを制御するソフトウェアである．しかし，後で詳しく説明するように，並行処理や実時間に関する処理が関連している．このような問題の解決策として，アスペクト指向技術，とくに MDSOC（Multi-Dementional Separation of Concerns）[8] の考えが提案されている．MDSOC では，ソフトウェアをアスペクトの集合とみて，それぞれのコンサーンを実現したものを合成し，ソフトウェアを作成できる．

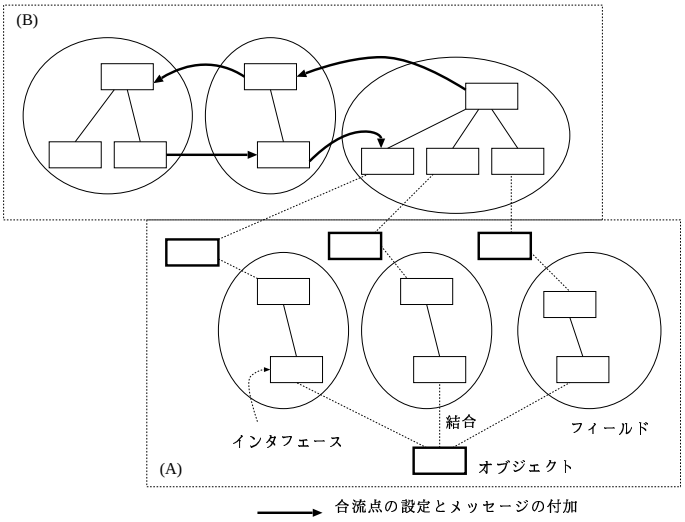

図 1　アスペクト指向ソフトウェアアーキテクチャスタイル

我々は，これらの考えを整理し，フィールドを提案する．フィールドは構成要素間の協調関係を表し，オブジェクトに付随する．フィールド内の構成要素はオブジェクトのインタフェースと表し，オブジェクトと結合する．移動するオブジェクトの場所（図 1(A) 参照）もアスペクト（図 1(B) 参照）もフィールドとして統一的にモデル化する．ソフトウェアはフィールドの集合だと考える．

我々は簡素であることは大きな利点があるとの認識を持ち，インタフェースとオブジェクトの結合，フィールドの合成は，合流点にメッセージを付加するというアスペクト指向が与える統一的な枠組みだけで整理する．インタフェースとオブジェクトの結合を考えると，インタフェースはそのオブジェクトがそのフィールドにいる場合のインタフェースを規定しているだけであり，実現自体はオブジェクトがしていることになる．つまり，インタフェースはメッセージをオブジェクトに委託している．これは，インタフェースのメッセージ受信を合流点として，このオブジェクトへのメッセージ通信を付加していると考えることができる．フィールド間の関係も同様に，各フィールドに存在するインタフェースは何か別のフィールドに存在するインタフェースのインタフェースを表している．これも前述の例のように記述できる．Smalltalk [3] がオブジェクトへのメッセージ通信だけですべてを記述することができるように，アスペクト指向の統一的な枠組みのなかで，MDSOC，ロールに基づく計算，場を用いた計算という異なる考えを統一的に記述できる．

3 自動販売機の金銭管理システムのアスペクト指向ソフトウェアアーキテクチャ

本節では，自動販売機の金銭管理システムを例にとり，アスペクト指向ソフトウェアアーキテクチャスタイルを適用し，そのアスペクト指向ソフトウェアアーキテクチャを構築する．

3.1 金銭管理システムの金銭

前述のように，自動販売機の金銭管理システムの金銭は自動販売機中を移動し，その場所によって振る舞いを変更する．図 2は自動販売機中の金銭の移動を表す．

(1) 投入された貨幣は正常な貨幣であるかの検査を受ける．
(2) 正常な貨幣であれば，商品購入のための代金として受け入れられる．
(3) つり銭が不足している場合，金銭はつり銭チューブに保管される．
(4) つり銭が十分な場合，金銭は金庫に保管される．

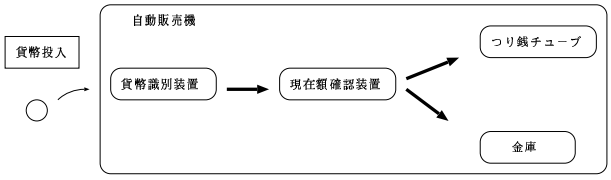

図 **2** 自動販売機中の金銭の移動

それぞれの場所における他のオブジェクトとの協調関係をフィールドとして実現すると，図 3のようになる．

– 受け入れフィールド
　　金銭は，貨幣識別装置によって，正常な貨幣であるかの検査を受ける．
　　この場合，貨幣識別装置にとっては，金銭は検査の対象物である．
– 販売フィールド

販売時には，現在額管理装置によって投入された代金として扱われる．

– 払い出しフィールド
  販売後，金銭はつり銭チューブに格納され，つり銭として扱われる．
– 格納フィールド
  十分なつり銭が用意されている場合には，金庫にて貯金として扱われる．

このように金銭は，各フィールドで異なる振る舞いをする．また，各フィールドの構成要素にとっては，金銭は金銭ではなく，それぞれ別のものとして扱われる．

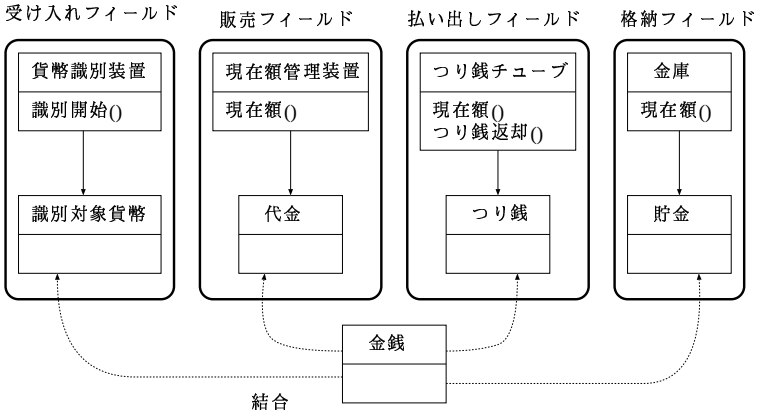

図 3　金銭に関するフィールド

3.2　金銭管理システムにおけるコンサーン

これまでの自動販売機制御ソフトウェア開発の経験 [7] から金銭管理システムを以下のコンサーンに分離した．

– コアコンサーン[1]（ハードウェア制御）
  組み込みソフトウェアはハードウェア制御を目的としたソフトウェアである．ハードウェアをオブジェクトとしてみた構造分割が自然であり，このコンサーンがコアコンサーンとなる．
– 並行処理
  各ハードウェアは並行に動作するので，ハードウェアをオブジェクトとした場合には，各オブジェクトに並行処理に関する記述が横断する．
– 実時間処理
  ハードウェアの動作の異常などは時間を計測して検知するしかなく，実時間処理が必要不可欠である．各ハードウェア間の協調関係を記述する場合，故障を検知するための実時間処理の記述が必要となる．

[1] コアコンサーンという用語は AspectJ [5] の用語である．AspectJ で合流点を設定する側をコアコンサーン，付加されたメッセージによって実現される側をアスペクトと呼ぶ．アスペクト指向の説明のさいには，一般的に用いられているので，本論文でもこれらの用語を用いる．

コアコンサーン（ハードウェア制御）

金銭管理システムは協調動作する複数のハードウェアを制御するソフトウェアである．金銭管理システムをオブジェクト指向分析する場合，ハードウェアの制御に着目して構造を整理する．この場合，ハードウェア制御がコアコンサーンとなる．金銭管理システムを分析し，以下のハードウェアの存在を確認した（図 2参照）．

- 貨幣識別装置
- 現在額管理装置
- つり銭チューブ
- 金庫

ハードウェアの追加，変更を考慮し，構造を分割した結果，図 4に示すフィールドを得た．

- ハードウェア毎に論理ハードウェアを用意
  論理ハードウェアを介してハードウェア制御を行うことで，ハードウェアの変更に強くなる．ハードウェアの変更は論理ハードウェアに局所化できる．図中の論理〇〇は論理ハードウェアである．
- 論理ハードウェアを管理するコントローラを用意
  コントローラを用意して，ハードウェア間の通信をなくすことで，各ハードウェアは独立に変更が可能である．図中の金銭管理システムがコントローラに相当する．

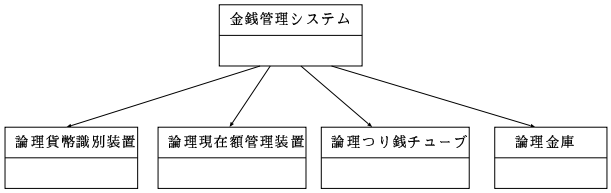

図 4　コアコンサーンフィールド

並行処理

並行処理に着目して組み込みソフトウェアの構造を分割すると，

- 管理するオブジェクト
- 管理されるオブジェクト

に分割できる（図 5参照）．管理されるオブジェクトとは並行オブジェクト（ScheduledObject）であり，管理するオブジェクトとはそれらを管理する疑似並行実行用のモニタ（Monitor）である．モニタでイベントを待ち行列で管理し，各オブジェクトに順に実行権を与え，イベントを通知する．

実時間処理

実時間に関する処理を分析した結果，図 6のフィールドが得られた．実時間処理は，時間を計るタイマ（Timer）とタイマを使用するオブジェクト（TimedObject）間のメッセージ通信で実現でき，これらのオブジェクト間には，つぎの関係がある．

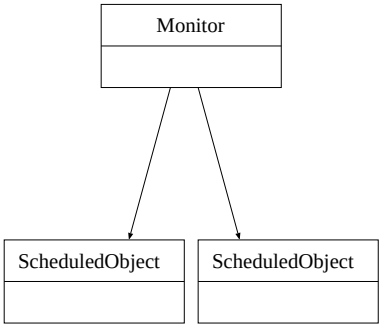

図 5　並行処理フィールド

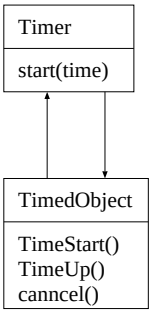

図 6　実時間処理フィールド

– TimedObject は処理の実行前に必ずタイマを起動する．
– タイマは時間の計測が終了したら，それを TimedObject に通知する．

これは，実時間処理の典型的なパターンである．

3.3　フィールドの合成

各フィールド間の関係は図 7のようになる．合流点にメッセージを付加するとき，コアコンサーン側のコード，または付加するメッセージ間の呼び出しの順序関係が現れる．図 7では説明のために，わかりやすい AspectJ の before，after，around の記述を用いている．before，after はコアコンサーン側のコードの前後を表し，around はコアコンサーン側のコードを付加するメッセージに置き換えることを意味している．

– 各論理ハードウェア，およびコントローラは並行オブジェクトとして扱いたいので，これらへのメッセージ通信は Monitor（並行処理フィールド）へのメッセージ通信とする．
– 各論理ハードウェアは Monitor から実行権が与えられ，動作する．ScheduledObject は Monitor から実行権が与えられると，論理ハードウェアにメッセージを委託する．
– 各論理ハードウェアは金銭に関係するフィールド中の構成要素と同一のものを表している．したがって，論理ハードウェアは対応する構成要素にメッセージを委託する．
– 貨幣識別装置は貨幣の識別時，時間を計測し，時間内に識別ができない場合，異常と判断する．したがって，貨幣識別装置は TimedObject（実時間処理フィールド）としても振る舞うので，TimedObject にメッセージを委託する．

これらの関係にしたがって各フィールドを合成すると，金銭管理システムを実現できる．

4　ソフトウェアアーキテクチャ群の構築

組み込みソフトウェアの応用領域は十分に大きいので，単一のソフトウェアアーキテクチャではなく，抽象度を鍵に階層化・整理したソフトウェアアーキテクチャ群を定義する．

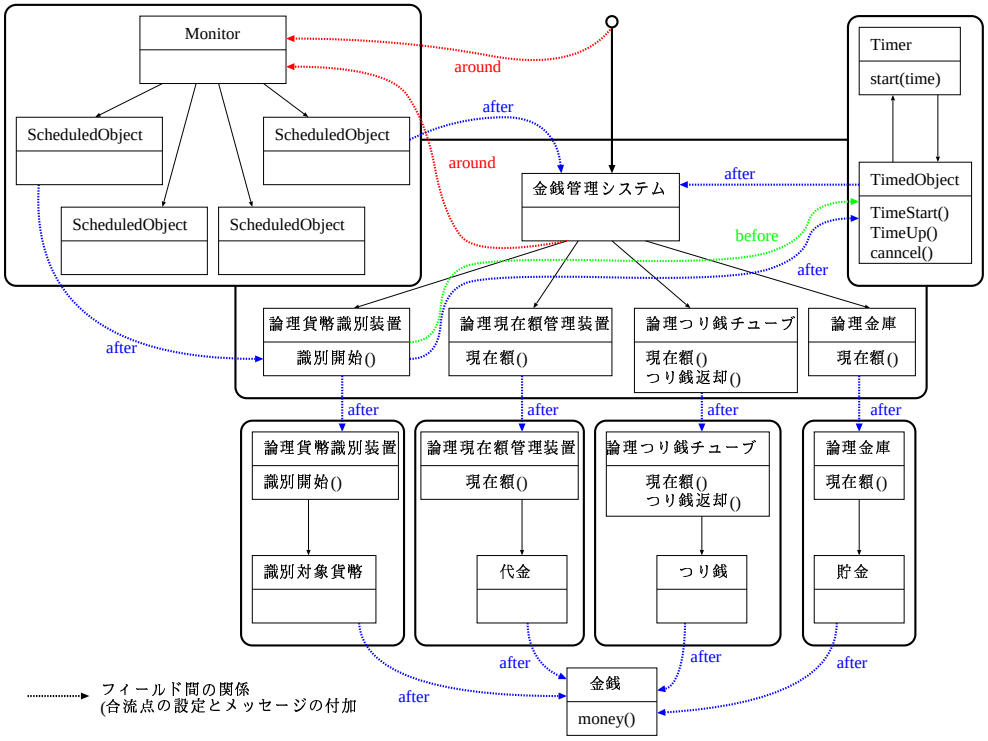

図 7　金銭管理システムのアスペクト指向ソフトウェアアーキテクチャ

4.1　抽象的な組み込みソフトウェアのソフトウェアアーキテクチャ

構築したソフトウェアアーキテクチャを抽象化すれば，一般的な組み込みソフトウェアのソフトウェアアーキテクチャが構築できる．

事例では，場所を移動することによって振る舞いを変化されるオブジェクトの分析を行った．しかし，3節の最初に述べたようにこのようなオブジェクトは事例毎に異なるので抽象化することは難しい．あまりに抽象的ではあるが，本ソフトウェアアーキテクチャでは，移動オブジェクトとする．

構築したアスペクト指向ソフトウェアアーキテクチャを分析した結果，以下の 3 つを組み込みソフトウェアに共通なコンサーンとした．

- コアコンサーン（ハードウェア制御）
- 並行処理
- 実時間処理

これらのコンサーンを実現したフィールドを抽象化すると図 8のようになる．これは一般的な組み込みソフトウェアのアスペクト指向ソフトウェアアーキテクチャである．組み込みソフトウェアの実現する場合，

- 場を移動するオブジェクトが何であるか
- ハードウェアは何であるか

を考慮し，このソフトウェアアーキテクチャを適用すればよい．

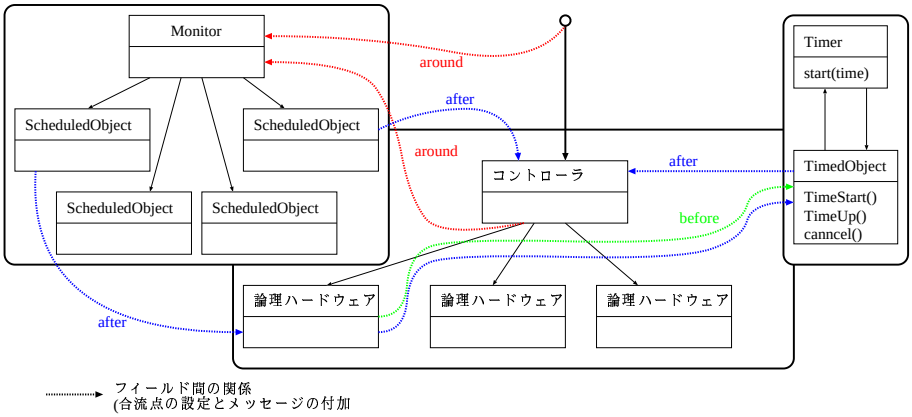

図 8　抽象的な組み込みソフトウェアのアスペクト指向ソフトウェアアーキテクチャ

4.2　ソフトウェアアーキテクチャの階層化

事例に対して構築したソフトウェアアーキテクチャと抽象的なソフトウェアアーキテクチャは抽象度を鍵に図 9のような階層関係を構築する．組み込みソフトウェアの応用領域は大きいので，単一のソフトウェアアーキテクチャですべての事例に対応することは難しい．このように，抽象化したソフトウェアアーキテクチャをもとに，複数の事例を適用し，分類することが重要である．

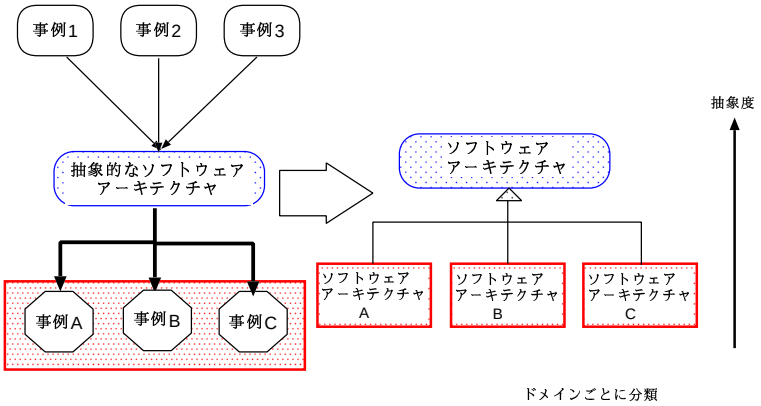

図 9　ソフトウェアアーキテクチャの階層

今後，多数の事例を積み重ねることで，より細かな分類が可能である．組み込みソフトウェアの開発時には，対象とするソフトウェアが階層のどの部分に位置するのかということを考慮する．似たような応用領域のソフトウェア作成時にはその応用領域のソフトウェアアーキテクチャを参考にすればよい．

5　議論

我々は，以下を本ソフトウェアアーキテクチャの利点と考える．

– 横断的に関連するコンサーンとして，並行処理，実時間処理を矛盾なく分離，記述できる．
– ソフトウェア中を移動するオブジェクトの場所の変化にともなう振る舞いの変化を場所毎に分離，記述できる．

提案した組み込みソフトウェアのアスペクト指向ソフトウェアアーキテクチャの構造について，それぞれの観点から，その妥当性について議論する．

5.1 コンサーンの分離について

我々はこれまでに自動販売機制御ソフトウェアのソフトウェアアーキテクチャスタイル [7] を提案した．自動販売機制御ソフトウェアを並行に動作する状態遷移機械の集まりとしてモデル化し，疑似並行実行用の疑似並行実行モニタを実現した．このさいの問題点は，並行処理の記述がハードウェアを制御するための状態遷移機械の記述に含まれていたことであり，並行処理に関する変更が状態遷移機械に影響を与えた．例えば，メッセージ送信に特急モードを追加する場合などには，状態遷移機械を変更しなければならなかった．この経験から我々は，並行処理を横断的に関連するコンサーンとして認識した．

実時間処理は，メッセージ送信の対象となるオブジェクトと時間を計るためのタイマオブジェクトに同時に非同期メッセージを送り，結果を同時待ち受けするという方法で実現していた．このことから，以下の点が確認できた．

– タイマオブジェクトは実現，実行の環境に依存すること（時間を計るための API が必要）
– 実時間処理は並行処理と密接に関連していること

以上のことから，実時間処理を横断的に関連するコンサーンとして考えた．

組み込みソフトウェアは抽象化すると，ハードウェア制御，並行処理，実時間処理によって記述できるとの仮定のもとに，これらの分離を試みた．これからの開発で，新たなコンサーンの分離の必要があるかもしれないが，これは今後の課題とする．

並行処理フィールドでは，疑似並行実行モニタの容易な実現と可搬性を得た．並行処理に関する記述を変更する場合には，疑似並行実行モニタの記述と，疑似並行実行モニタへのメッセージ通信の記述だけを変更すればよい．疑似並行実行モニタへのメッセージ通信は，合流点に付加されたメッセージなので，この変更はコアコンサーン（ハードウェア制御），実時間処理のコードには変更を与えない．例えば，メッセージ通信に特急モードを追加する場合，疑似並行実行モニタと合流点の記述を変更すればよい（図 10参照）．並行オブジェクトに優先度を与え，スケジューリングポリシーを変更する場合には `Monitor` と `ScheduledObject`（並行処理フィールド）を変更すればよい（図 11参照）．両者ともに，コアコンサーン（ハードウェア制御），実時間処理のコードには影響しない．

実時間処理に関する問題は，並行処理と絡み合った問題であり．実時間処理フィールドでは，これらを分離して記述することを試みた．その結果，処理を実行する前にタイマを始動し，タイマからの応答を待ち受けるという典型的なパターンを実時間処理として分離記述することに成功した（3.2節参照）．これらの記述は，アスペクト間の関係として記述されているので，コアコンサーン（ハードウェア制御），並行処理の記述とは独立である．例えば，つり銭チューブの故障を検知するために，

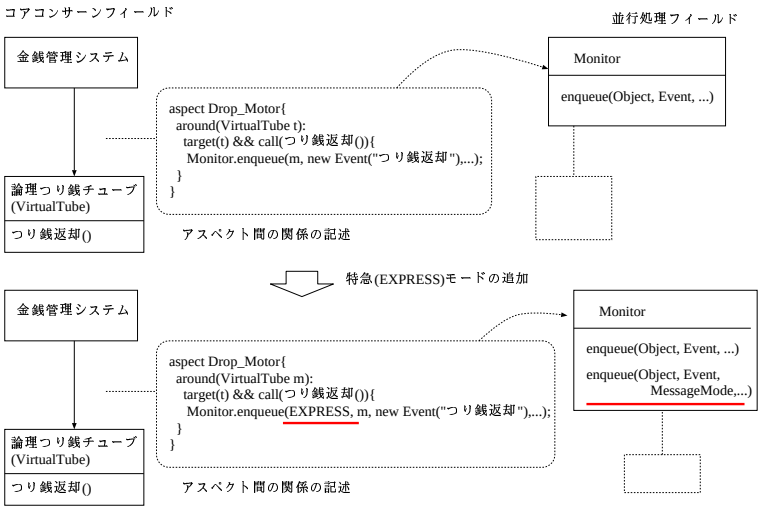

図 10　特急モードの追加

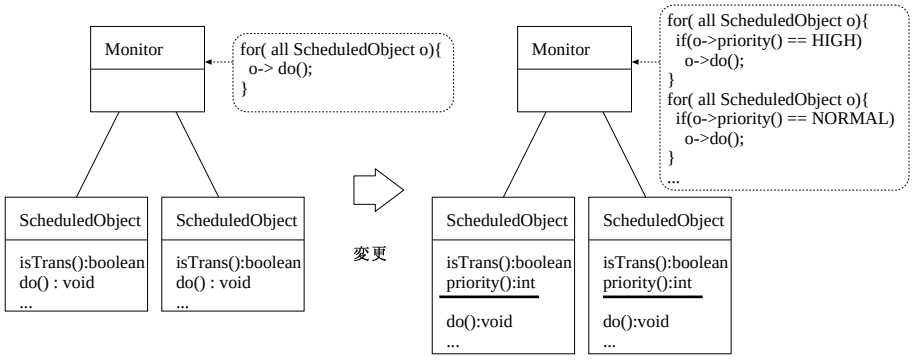

図 11　優先度の追加

つり銭チューブからの応答時間を計測したい場合，図 12のように合流点の記述を追加するだけでよい．

5.2　移動オブジェクトの分離について

移動オブジェクトの実現方法としてフィールドを用いる方法以外に以下の方法が考えられる．

- 状態遷移機械として実現する
- 多相型として実現する

前者の場合，金銭を場所の変化にともなって，振る舞いを変化させる状態遷移機械として実現する．この場合，組み込みソフトウェアは状態遷移機械の集まりとして実現されていることから金銭に関係するハードウェアはすべてが金銭自体と密接に関係し，ハードウェアの追加，変更が金銭に影響を与える．また，金銭の変更がハードウェアに影響を与える．例えば，金銭管理システムの金銭の場合，図 13のような振る舞いの変化を考える．これに代金を返却するための代金返却装置の追加を

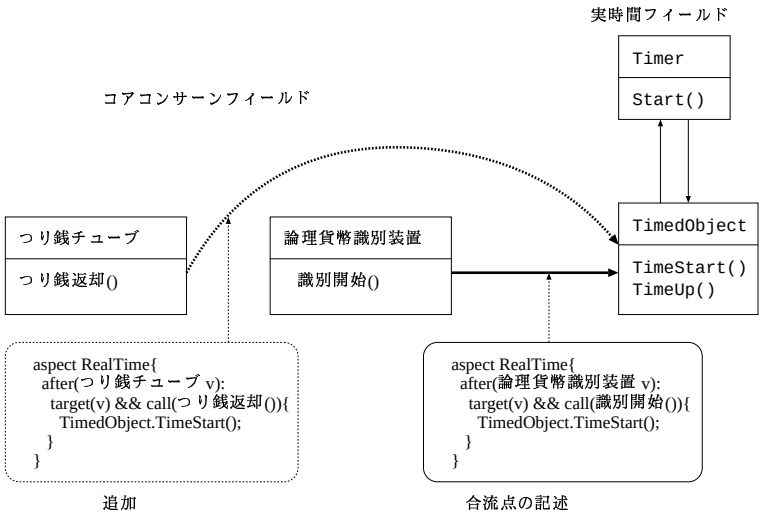

図 12　実時間処理の追加

考えると，図 13破線のように遷移を変更しなければならない．つまり，ハードウェアの追加が金銭の実現に影響を与えている．

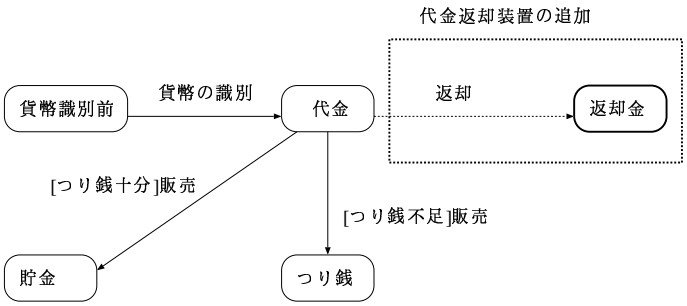

図 13　金銭の変化

フィールドを用いた場合には，

- コアコンサーンフィールドに論理ハードウェアを追加し，
- 返却のためのフィールドを追加するだけでよい（図 14参照）．

代金返却装置と関係するのは，金銭ではなく，返却金なので，金銭には影響を与えない．

後者の場合，金銭をスーパークラスとして，場所毎の振る舞いをサブクラスに実現できる．しかし，場所の変更にともなって適当なインスタンスを生成する必要があり，これを実現するためには，各クラスにクローンを生成するためのメソッド [2] を用意しなくてはならない．この場合，コードの可読性が悪くなるという問題がおきる．

このように振る舞いを変更するオブジェクトは，その振る舞い毎に関係するオブジェクトが異なる．我々は，振る舞い毎の他のオブジェクトとの協調関係をフィー

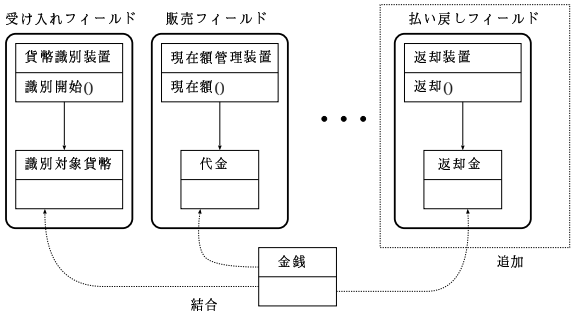

図 14　返却装置の追加

ルドを用いて分離，記述している．移動オブジェクトはその性質上，多くのハードウェアと関連するので，それぞれの関連毎にフィールドとして分離することで，ハードウェアとの関連を弱くしている．

6　おわりに

本研究では，組み込みソフトウェアの階層化したアスペクト指向ソフトウェアアーキテクチャ群を構築した．構築したソフトウェアアーキテクチャにしたがえば，

- 場を移動するオブジェクトを自然にモデル化できること，
- 実時間性，並行処理を矛盾なく記述できること，
- これらをフィールドとして統一的に記述できること，

を示した．提案したアスペクト指向ソフトウェアアーキテクチャスタイルを他の応用領域に適用し，考えに矛盾がないことを確認することを今後の課題とする．

参考文献

[1] C. Constantinides, A. Bader, T. Elrad and M. Fayad "Designing an Aspect-Oriented Framework in an Object-Oriented Environment," *Computing Surveys*, 32, 41, 2000.
[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, 1994.
[3] A. Goldberg, *Smalltalk-80: the interactive programming environment*. Addison-Wesley, 1984
[4] D. Garlan, and D. E. Perry, "Introduction to the Special Issue on Software Architecture," *IEEE Transaction on Software Enginnering*, Vol. 21, No. 4, pp.269-274, 1995.
[5] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W.G. Griswold,"An Overview of AspectJ", in *Proceedings of the European Conference on Object-Oriented Programming(ECOOP)*, Springer-Verlag. Hungary, 2001.
[6] 熊崎 敦司，後藤 修平，野呂 昌満，張 漢明，"組み込みソフトウェアのアスペクト指向ソフトウェアアーキテクチャ～自動販売機のカップ機構制御ソフトウェアアーキテクチャの構築～", ソフトウェア技術者協会 ソフトウェアシンポジウム, 2003.
[7] 森 貴彦，服部 和真，"自動販売機制御ソフトウェアのアーキテクチャスタイルに関する研究"，南山大学 経営学生論集，第 *16* 集，2002.
[8] H. Ossher and P. Tarr, "Multi-Dimensional Separation of Concerns and the Hyperspace approach", in *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development.* Kluwer, 2001.
[9] D. Schmidt, M. Stal, H. Rohnert, F. Buschman, *Pattern-Oritented Software Architecture , Volume 2: Patterns for Concurrent and Networked Objects*, John Wiley & Sons, 2000
[10] T. Tamai, "Evolvable Programming based on Collaboration-Field and Role Model," *International Workshop on Principles of Software Evolution,* 2002.
[11] P. Tarr, H. Ossher, W. Harrison and S.M. Sutton, "N Degrees of Separation: Multi-Demensional Separation of Concerns", in *Proceedings of the 21st International Conference on Software Engineering*, May. 1999.
[12] "Flage ホームページ，" http://www.ipa.go.jp/NEWSOFT/public/Flage/flage_start.html .