アスペクト指向アーキテクチャに基づく組込みソフトウェアの設計法の提案

野呂 昌満[1,a)]　沢田 篤史[1,b)]　張 漢明[1,c)]　繁田 雅信[2]

概要：本研究では，ハードウェア技術の進歩により複雑，高機能化する組込みシステムのためのソフトウェア設計法を提案する．多種類のセンサやアクチュエータから構成される組込みシステムでは，システムの外部環境と内部状態に応じた複雑な制御が求められる．並行で非同期に動作する部品を統合し，不具合のないソフトウェアによる制御を実現するために，アスペクト指向ソフトウェアアーキテクチャに基づく設計法を提案する．提案設計法の基礎となるソフトウェアアーキテクチャは，組込みシステム固有の関心事をアスペクトとして分離することで，凝集度 (強度) が高く結合度の低いモジュール設計を可能とする．本設計法の特長の一つは，アスペクトモジュールの振舞いを形式的に定義し，それに基づき共有資源を特定することにある．さらに, 共有資源上の排他制御論理の実現プロセスを含む, アーキテクチャ中心設計プロセスを提案する．実用に供する金銭授受装置を題材にした事例を用いて提案設計法の有用性と妥当性を示す．

1. はじめに

近年，センサ，アクチュエータデバイスの多様化とマイクロプロセッサの高性能化により，組込みシステムにはより多くの種類の部品が使用され，高い機能が実現されるようになってきた．結果として, 組込みソフトウェアには, 多様な部品を協調させる複雑な制御の実現が求められるようになった．従来の組込みシステム開発では, 並行に動作するサブシステムやプロセスとその同期方式の詳細を決定してからそれぞれの制御機能を実現してきた．組込みシステムの質変化に伴い, この従来開発法はもはや適切ではなくなってきている．昨今の組込みシステムは外部環境や内部状態に応じて，稼働させるセンサやアクチュエータの種類や数，情報交換方法が変化するという特徴がある．さらに，システムが置かれた状況毎に協調させる部品や協調の方法が異なり，共有資源の特定や同期方式の決定も単純な問題ではなくなってきている．以上のような組込みシステムの変化に対処可能な作成方法が求められている [1].

本研究の目的は，ソフトウェアアーキテクチャに基づく組込みシステムのためのソフトウェア設計法を確立することである．我々はこれまでに，組込みシステムに対する典型的な関心事をアスペクトとして分離したソフトウェアアーキテクチャを設計してきた [2]．さらに，このアーキテクチャに基づくことで組込みソフトウェア開発プロセスをどのように編成できるかについての基本的なアイデアを示してきた [3]．本稿では，このアイデアを設計法として整理するとともに，簡単な事例への適用を通じて妥当性を評価・確認することで，目的の達成を図る．

提案する設計法では，ソフトウェアアーキテクチャが規定するモジュールの形式的な振舞い仕様記述に基づく共有資源の特定と，排他制御論理の設計のためのプロセスを定義した．これを用いることで，凝集度が高いモジュールの，結合度が低い集合としての設計が可能になる．さらに，並行・非同期処理において懸念されるデッドロック等の不具合の発生を実行前に排除する設計も可能となる．実際の金銭授受装置に使用される紙幣搬送システムを模した組込みシステムのためのソフトウェアを設計することで，本設計法の有用性，妥当性を示す．

2. 複雑・多様化する組込みシステムのためのソフトウェア設計

組込みシステムは一般に，センサやアクチュエータなどシステムの用途に応じたハードウェア部品をマイクロプロセッサに接続した構成をとる．組込みシステムを構成するセンサはシステム外部環境の状態を取得し，アクチュエータは外部環境の状態に影響を与える．組込みソフトウェアはこれらセンサ，アクチュエータを含む部品を制御し，シ

[1] 南山大学
Nanzan University, Showa, Nagoya 466-8673, Japan
[2] 富士電機株式会社
Fuji Electric Co., Ltd.
[a)] yoshie@nanzan-u.ac.jp
[b)] sawada@nanzan-u.ac.jp
[c)] chang@nanzan-u.ac.jp

ステムに求められる機能を実現する．マイクロプロセッサ上で動作する組込みソフトウェアの設計では，センサやアクチュエータを用いてシステムに求められる機能を実現する方式の他に，並行かつ非同期に発生する外部環境の事象を適切に取り扱う方式を明確化することが求められる．

これまでの組込みシステムには比較的単純な機能のみが要求されてきた．そのような組込みシステムのためのソフトウェアの設計では，複雑な機能を実現することよりも，並行動作するプロセスの同期をとり，プロセス間の通信を適切に実現することが重視されてきた．すなわち，並行動作するプロセスを特定した上で，それらの間の通信手段や同期方式の設計をした後に，機能を設計する方法が一般的に取られてきた．

近年，センサやアクチュエータデバイスが多様化し，プロセッサも高性能化したことで，組込みシステムにはこれまでより多くの種類のハードウェア部品が使用され，多くの機能が実現されるようになった．結果として，それらを制御する組込みソフトウェアには，多種多様なハードウェア部品を協調させる複雑な制御の実現が求められるようになった．一方で，組込みシステムに特有のリアルタイム性，耐故障性，信頼性などの非機能特性に対する要求は相変わらず存在する．すなわち, 厳しい非機能要求を満たすように複雑な機能を実現しながら，並行，非同期システムとしての組込みシステムを制御する論理を実現することが現行の組込みソフトウェア開発には求められている．

以上をまとめると，並列処理問題を先に解決し，それを確定した上で非機能特性を分離するというアプローチは昨今機能しづらくなってきたと考えられる．すなわち、通信手段や同期方式の設計後に，並行動作する各部品の機能を実現するという従来の組込みソフトウェア設計法は必ずしも有効に機能しなくなってきたと考えられる．なぜなら，通信方式の選択は非機能特性である実行効率と密接に関係し，その選択は非機能特性を支配すると考えられることによる．一般化すると，アスペクトの付加や排除は非機能特性の軽重と完全に独立しえないものといえる．現行の開発においては,

- システムの外部環境や内部状態に応じて，稼働させるセンサやアクチュエータの種類，数が変化する，
- 共有する情報の種類や通信方式が変化する，など，

システムが置かれた状況毎に協調させる部品や協調の方法が異なる．すなわち，組込みソフトウェアの設計においては, 共有資源の特定と同期方式の決定が単純な問題ではなくなってきた．他方，このような状況においても，通信方式と非機能特性の選択を互換的に行うことは系統的な設計の観点から望ましくない．我々は，共有資源の特定とその実現を切り離すことで多様な非機能要求の実現に応えることとした．すなわち，共有資源の同定方法を明確にし，その豊富な実現方法を提供することで，同期問題と非機能特性の系統的な取り扱いを可能とする．

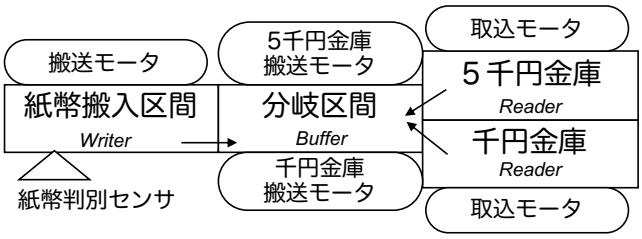

図 1 紙幣搬送システムの構成

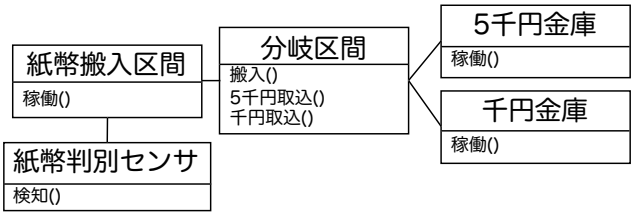

図 2 紙幣搬送システムの静的構造

図 1 に示す紙幣搬送システムを制御するソフトウェアの設計を考える．このソフトウェアが制御するハードウェアの仕様を以下に示す．

- 紙幣の搬送路として紙幣搬入区間と分岐区間，紙幣が格納される金庫として 5 千円金庫と千円金庫，紙幣の種類を判別する紙幣判別センサによる構成をとる．
- 紙幣は紙幣搬入区間に投入され，搬送モータによって分岐区間に紙幣が送られるように動作する．
- 紙幣が投入されると，紙幣判別センサはその種類を判別する．
- 各種金庫の取込モータは，合流区間の紙幣を金庫に取り込むように動作する．
- 分岐区間にはそれぞれの金庫に紙幣を送るためのモータが備わっている．
- 以上の構成要素は独立して設計，実装されており，組み合わせることでシステムが構築される．

図 2 に示すように，紙幣搬入区間，分岐区間，投入された紙幣の種類に応じた金庫間で，協調のための同期が行なわれる．

- 分岐区間には，搬入，5 千円取込，千円取込の操作が定義されている．
- 搬入操作には分岐区間に紙幣を置くためのモータの動作が定義され，5 千円取込操作，千円取込操作にはそれぞれ，対応する金庫に分岐区間上の紙幣を送るためのモータの動作が定義される．
- これらの操作が適切な順番で実行されることで，紙幣が金庫に搬送される．
- 5 千円札が投入されたさいには，搬入，5 千円取込の順に実行可能となる．
- 千円札が投入されたさいには，搬入，千円取込の順に実行可能となる．

紙幣搬送システムでは，投入された紙幣の種類に応じて，共有資源である分岐区間に対して操作を行なう部品が変化する．紙幣搬入のための分岐区間が増えた場合，共有資源となる分岐区間も状況に応じて変化する．分岐区間と紙幣の種類ごとに操作する部品が変わることから，可能な組み合わせそれぞれについて共有資源を特定し，排他制御のための方式を決定しなくてはならない．これらの設計が場当たり的に行なわれると，当然，ソフトウェアの信頼性は低くなり，保守性は悪くなる．

明確なモジュール分割の指針に従うことで信頼性，保守性の高いソフトウェアの設計を可能とする点で，ソフトウェアアーキテクチャに基づいて開発を行なう価値は高い [5]．上述の通り，組込みソフトウェアには，開発対象システムの機能を，非機能要求を満たしながら提供することが求められる．加えて，並行して動作するハードウェア部品を適切に制御しながら状況に応じた機能構成や通信方式の変更にも対応しなければならない．

我々はこれまでに，コンテキスト指向およびアスペクト指向を適用した組込みシステムのためのアーキテクチャ [2] を提案してきた．このアーキテクチャでは，並行に動作する状態遷移機械をモジュールの構成単位とし，組込みシステムに特有のリアルタイム性や耐故障性などの非機能要求に関する処理を，システムの機能を実現するモジュールに横断するアスペクトモジュールとして分離している．さらに，コンテキスト指向プログラミング [4] の概念を適用することで，振舞いを変化させる状況をコンテキストとして捉え，コンテキストに応じた振舞いや構成変更をそれぞれ独立したモジュールとして分離している．このアーキテクチャに基づく開発では，機能，非機能，コンテキストをそれぞれ分離することで，モジュールの独立性を高め（凝集度が高いモジュールの結合度が低い集合としての設計），柔軟性ならびに信頼性の高いシステムの設計が可能となる．

他方，モジュール間通信における同期や排他制御の設計については支援が必要である．アーキテクチャに基づかない場合でもこの事情は同様である．複雑で多様な機能が求められる組込みシステムでは，多くのモジュールが状況に応じて振舞いや通信方式を変化させることが必要になる．このことから，共有資源の特定と排他制御方式の設計のための体系的な方式が必要となっている．

以上，まとめると，本研究における技術課題は，つぎに示す二つの設計プロセスを明らかにすることである．

1) アーキテクチャに基づいてモジュールを設計するためのプロセス
2) モジュール間の共有資源を特定し，排他制御を設計するためのプロセス

3. 組込みソフトウェアのための設計プロセス

前節でまとめたように，本研究では，組込みシステムの設計法として，

- ソフトウェアアーキテクチャに基づく設計プロセスと，
- モジュール間の共有資源を特定し，排他制御を設計するプロセス

を明らかにすることを目的とする．

ここで提案する設計プロセスは，アーキテクチャ中心開発（Architecture-Centric Development）を基礎とする．提案するアーキテクチャにおいては組み込みソフトウェアとして必要十分な非機能特性が横断的関心事として分離されている．

我々は，これまでの幾多の開発経験から，組み込みソフトウェアにおいては，特に実時間性並びに並行処理が最重要であるとの着想を得た．並行処理アスペクトにおいては，共有資源の取り扱いが最重要課題であり，その取り扱いの適切性がソフトウェアの品質を支配する．他方，並行処理と実時間性は密接に関係する特性同士であり，両者に対する要求を十分に満足しなければならない．ここでは，共有資源の系統的な同定の上に立つ実時間処理の独立かつ有用な設計方法を提案する．

3.1 組込みシステムのためのソフトウェアアーキテクチャ

我々がこれまでに設計してきた組込みシステムのためのソフトウェアアーキテクチャ [2] を図 3 に示す．本アーキテクチャは，コンテキスト指向ならびにアスペクト指向を適用したものである．コンテキストに応じて機能や構成の切り替えを行なうための処理と非機能要求に関連する処理とを，システムに求められる機能を実現する処理から分離した構造を持つ．

図の中央にあるハードウェア（HW）およびそのサブクラスが，組込みソフトウェアの制御対象であり，システムに要求される機能を実現するためのコンポーネントである．これら仮想ハードウェアコンポーネントを囲む丸枠は，横断的に関連するアスペクトを示す．アスペクトとして，コンテキスト処理（Context），非機能要求に関して，並行性（Concurrency），実時間性（RealTime），耐故障性（FaultTolerant：図中では一部 F.T. と表記）を定義している．

これらアスペクトの構造は，引き出し線（破線）により関連付けられた枠内に示されている．それぞれの構造は，自己適応のための PBR（Policy-Based Reconfiguration）パターン [6] を適用して設計する．いずれのコンポーネントも，定義したポリシに基づき，仮想ハードウェアコンポーネントの状況や状態に適応し，静的あるいは動的に処理の再構成（織り込みを含む）を行なう構造となっている．

3.2 アーキテクチャに基づく設計プロセス

前述のソフトウェアアーキテクチャに基づく組込みソフトウェア設計法を図 4 に示す．提案設計法はつぎの段階か

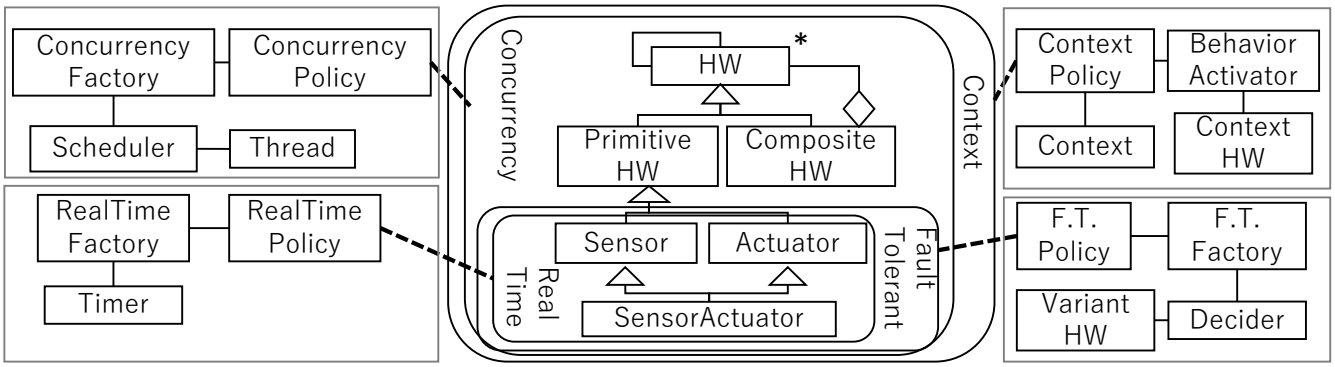

図 3 組込みシステムのためのアスペクト指向アーキテクチャ

ら構成される．

1. 仮想ハードウェアコンポーネントおよびアスペクトの抽出
2. 構成要素（コンポーネントまたはアスペクト）のインタフェース定義と協調関係の定義
3. 各構成要素の詳細設計
   a. 仮想ハードウェアコンポーネントの詳細設計
   b. コンテキストアスペクトの詳細設計
   c. 実時間性アスペクトの詳細設計
   d. 並行性アスペクトの詳細設計
   e. 耐故障性アスペクトの詳細設計
4. 構成要素間の通信と排他制御方式の詳細設計

段階 1 から 3 では，前節のアーキテクチャに従い，システムに要求される機能や非機能を実現する構成要素の詳細設計までを行なう．その後，段階 4 で構成要素間通信において共有資源を特定し，排他制御方式を決定する．

このような手順とした理由は，多数で多様なハードウェアデバイスを制御し，外部環境や内部状態に応じて求められる振舞いが異なるような組込みシステムの開発に対応することに起因する．すなわち，並行，非同期に動作する処理プロセスが状況によって異なる振舞いをするので，コンポーネント間の同期や排他制御の方式を，システムの全体構造を定義する段階では簡単に決定できない．実際に組込みソフトウェアの開発現場では，対象のハードウェアの動作仕様にそった制御を実現するために，状態遷移設計のさいに，同期イベントの待受け状態を導入することで，各コンポーネントに分散させる実現を行っている．これにより，暗に排他制御の論理を記述している事例が多く見受けられた．本節では以下，ぞれぞれの段階について説明する．ただし，段階 4 については次節で詳述する．

段階 1 では，組込みシステムに対する機能要求とハードウェア構成，さらに，システムに対する非機能要求に基づいて，仮想ハードウェアコンポーネントの構造とそれに横断的に関連するアスペクトの構造を決定する．ここでは，前節に説明したアーキテクチャに従い，システムに要求されるアスペクトのみを仮想ハードウェアコンポーネントに関連付ける．

段階 2 では，段階 1 で特定した各構成要素のインタフェースを定義するとともに，システムのユースケースを検討することで，並行プロセスとして協調動作する構成要素群の特定を行なう．

段階 3 では，段階 1 で特定した各構成要素内部の詳細設計を行なう．仮想ハードウェアコンポーネント，各アスペクトいずれも，イベントに対する状態遷移とアクションを定義することで，振舞いの詳細設計を行なう．前節のアーキテクチャでは，各モジュールが独立して定義されていることから，段階 3.a〜3.e は基本的に並行して行なうことができる．ただし，段階 3.c の実時間性アスペクトは，ハードウェア制御のための処理が実時間制約を守ることを保証するための構成要素であることから，仮想ハードウェアコンポーネントにおいてアクションが設計された後でなければ設計することができない．また，段階 3.e の耐故障性アスペクトは，ハードウェア制御処理において，冗長化やリカバリなどの耐故障処理を実現する役割を担うことから，同様に仮想ハードウェアコンポーネントのアクション設計後の実施となる．

3.3 共有資源と特定と排他制御方式の設計プロセス

構成要素間の通信と排他制御方式の詳細設計（段階 4）のプロセスはつぎのとおりである．

4.1 CSP によるモジュールの振舞いのモデル化
4.2 同期イベントに着目した共有資源の特定
4.3 共有資源内での排他制御論理の定義

前節で説明した段階 3 までに記述された各コンポーネントの状態遷移仕様には，制御対象であるハードウェア等の動作仕様を加味して，同期待ち等の状態が導入されている．このようなモジュール間の協調の論理を形式的に記述するために，イベント授受に着目する．段階 4.1 では，各構成

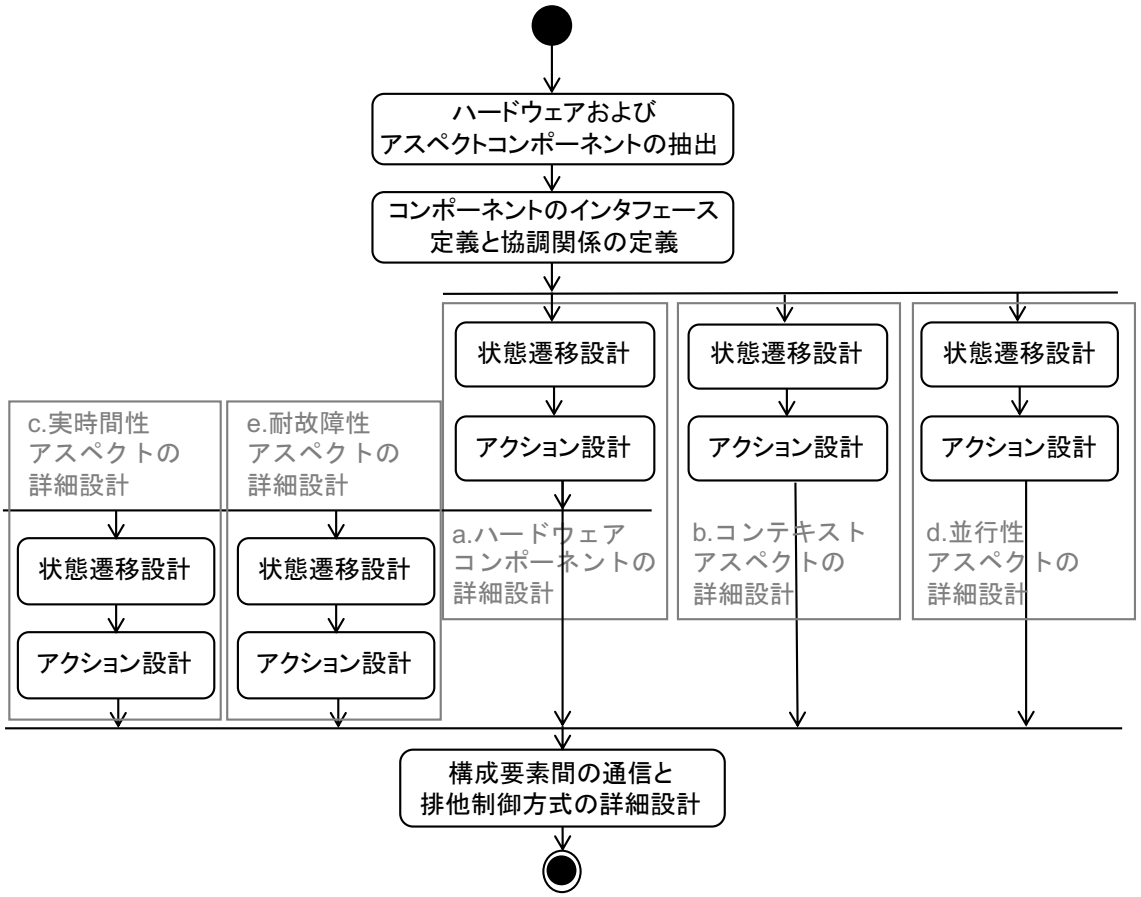

図 4 アーキテクチャに基づく開発プロセス

要素に対して設計された状態遷移仕様をイベント授受による振舞いの記述に変換する．記述には，代表的なプロセス代数理論である CSP を用いる．

段階 4.2 は，段階 4.1 で記述したイベント授受の振舞い記述の中から，同期イベントに関連する記述に着目することで，共有資源を特定する．CSP 記述におけるプロセスを特定構成要素の振舞いを表現するものとして捉えれば，同期イベントが構成要素間の通信を表す．他のプロセスとの同期を取るプロセスを識別することで，共有資源を特定することができる．

共有資源を特定するさいには，同じプロセス間での連続した同期イベントが，特定の共有資源への排他制御を特定する手掛かりになる．すなわち，図 5 に示す CSP の記述パターンが現れたとき，同期イベントはプロセス相互に他方のイベント発生を待つことを意味する．各モジュールの振舞い記述の中から，このような記述パターンを発見することで，共有資源への排他制御を特定することができる．

段階 4.3 では，共有資源に対する操作の実行順序を順路式 [7] 同等の記述を CSP を用いて行い，共有資源上での排他制御を設計する．共有資源への操作の実行順序を制限する順路式同等の記述を行うことで構成要素間の同期 (とくに相互排除) を実現する．記述された各モジュールの振舞いと排他制御論理との整合性を FDR[8] 等のモデル検査器

```
Process1 = ...−>syncA−>syncB−>...
Process2 = syncA−> ... −>syncB−>...
System = Process1
         [| { syncA, syncB } |]
         Process2
```

図 5 CSP 記述パターン

を用いて検証した後に，排他制御の論理を CSP 記述に従って詳細設計する．

4. 事例検証

本章では，紙幣搬送システムを事例として，アーキテクチャに基づく形式手法を用いた設計法の有用性について述べる．

4.1 紙幣搬送システムの設計

図 6 は，前述のアーキテクチャに基づいて設計した紙幣搬送システムの静的構造である．ここでは，コンテキストに応じたコンポーネントの振舞いについて議論したいので，仮想ハードウェアコンポーネントおよびコンテキストアスペクトによって規定される構造だけを示す．

紙幣搬送システム（ChangingMachine）の構成要素とし

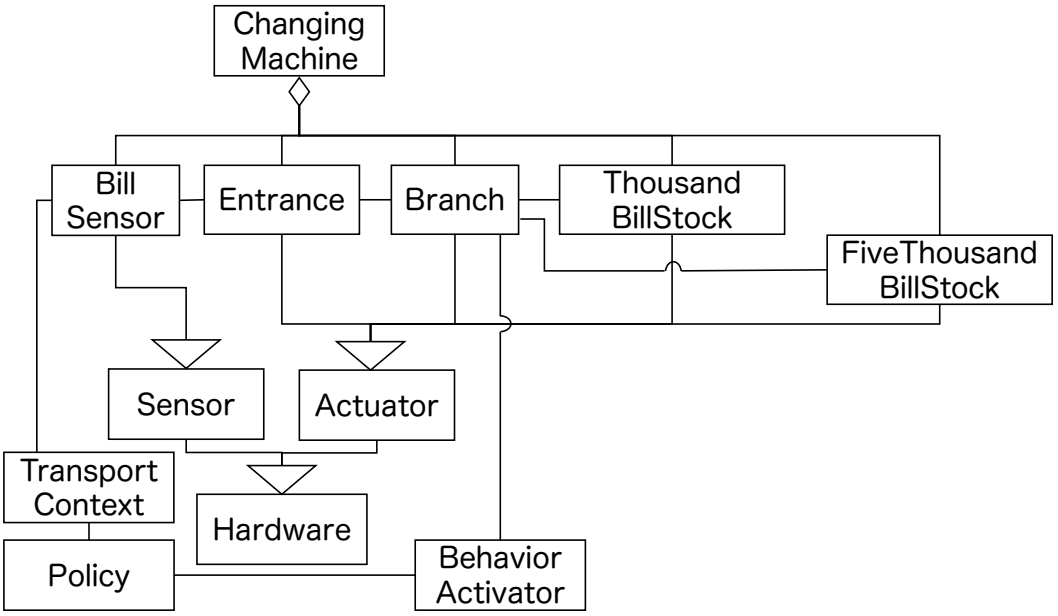

図 6 紙幣搬送システムの静的構造の詳細

て, 紙幣判別センサ（BillSensor), 紙幣搬入区間（Entrance), 分岐区間（Branch), 金庫（FiveThousandBillStock, ThousandBillStock）があり，それぞれに対する仮想ハードウェアコンポーネントが定義されている．これらのコンポーネント群では紙幣の種類をコンテキストとして搬送方法を決定する．搬送コンテキスト（TransportContext）が紙幣判別センサが検知する紙幣の種類を受け，ポリシ（Policy）が振舞い活性器（BehaviorActivator）によりハードウェアコンポーネント群を再構成する構造として定義している．コンテキストに応じた再構成の対象は分岐区間であり，これが変更されると，紙幣の保管先金庫が変わるようになっている．

それぞれのハードウェアコンポーネントのインタフェースを表 1 のように設計する．紙幣搬入区間は, 分岐区間に紙幣を送るために, put イベントを送ることでそれぞれの区間のモータが協調する．各金庫 (FiveThousandBillStock, ThousandBillStock) は，分岐区間から紙幣を取り込むために，pull5000, pull1000 メッセージを送る．紙幣判別センサは，ハードウェアデバイスから 5 千円札，千円札検知のイベント (detect5000, detect1000) を受け，これをきっかけとして，コンテキストコンポーネントが動作し，分岐区間での紙幣搬送先金庫が決定される．

同様にコンテキストアスペクトのインタフェースを表 2 に示す．コンテキストアスペクトでは，搬送コンテキストが紙幣判別センサからの紙幣検知イベント（notify5000, notify1000）を受け，検知した紙幣の種類が変化を検知すると，ポリシに再構成イベント（eval）を送り，振舞い活性器が変更された紙幣の種類に応じた経路を活性化（activateBranch5000, activateBranch1000）する．

図 7，8 は，それぞれ，仮想ハードウェアコンポーネント

表 1 ハードウェアコンポーネントのインタフェース

モジュール	イベント	振舞い
紙幣搬入区間 (Entrance)	run	紙幣を搬送
分岐区間 (Branch)	put	紙幣を配置
	pull5000	5 千円金庫に紙幣取込
	pull1000	千円金庫に紙幣取込
5 千円金庫 (FiveThousand BillStock)	run	紙幣取込
千円金庫 (Thousand BillStock)	run	紙幣取込
紙幣判別センサ (BillSensor)	detect5000	5 千円札検出
	detect1000	千円札検出

表 2 コンテキストアスペクトのインタフェース

モジュール	イベント	振舞い
搬送コンテキスト (Transport Context)	notify5000	紙幣種の変化をポリシに通知
	nofify1000	紙幣種の変化をポリシに通知
ポリシ (Policy)	eval	コンテキストに応じた再構成を実行
振舞い活性器 (Behavior Activator)	activate Branch5000	五千円金庫に搬送する経路を活性化
	activate Branch1000	千円金庫に搬送する経路を活性化

およびコンテキストアスペクトのイベント授受の順序関係を整理し，状態遷移機械として設計したものである．これらの状態遷移設計の後に，各構成要素がイベントを受けたさいのアクションが設計される．

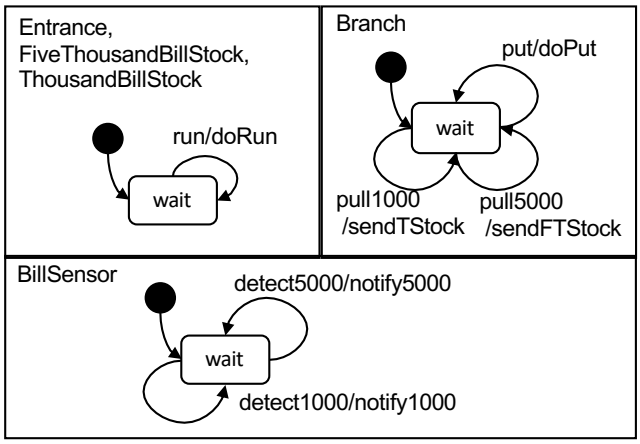

図 7 仮想ハードウェアコンポーネントの状態遷移設計

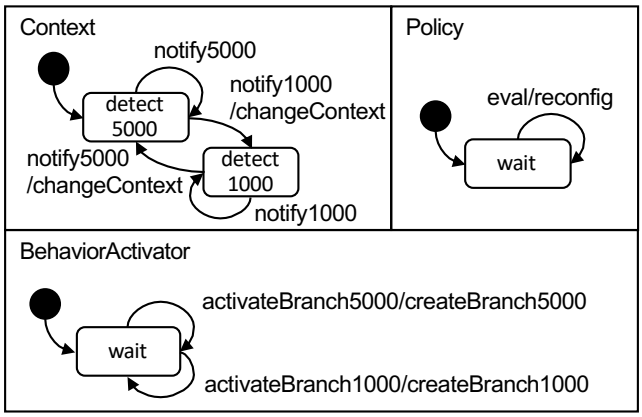

図 8 コンテキストアスペクトの状態遷移設計

```
Entrance
    = put->sync1->sync2->Entrance
Branch
    = put->Branch
      [] detect5000->pull5000->Branch
      [] detect1000->pull1000->Branch
FiveThousandBillStock
    = sync1->pull5000
      ->sync2->FiveThousandBillStock
ThousandBillStock
    = sync1->pull1000
      ->sync2->ThousandBillStock
BillSensor
    = detect5000->BillSensor
      [] detect1000->BillSensor

System
    = Branch
      [| msg |]
      ( Entrance
        [| sync |]
        ( FiveThousandBillStock
          ||| ThousandBillStock )
        ||| BillSensor )
msg = { put, pull5000, pull1000,
        detect5000, detect1000 }
sync = { sync1, sync2 }
```

図 9 紙幣搬送システムの構成要素の振舞い

4.2 共有資源の特定と排他制御の設計

共有資源の特定と排他制御論理の設計プロセスでは，前節で設計した各構成要素の状態遷移仕様に基づいて，その振舞いを CSP で記述する．

ここでは，仮想ハードウェアコンポーネント間の共有資源の特定を例に説明する．図 9 に，各仮想ハードウェアコンポーネントの状態遷移仕様から得られる記述を示す．put，pull5000，pull1000 は構成要素間の通信を表す同期イベントである．detect5000，detect1000 は紙幣を検知し，紙幣の種類をコンテキストとして分岐区間の振舞いが変化することを表す同期イベントである．sync1，sync2 は排他制御のための同期イベントである．

ここで，紙幣搬入区間の振舞いを表すプロセス Entrance において，図 5 に示した排他制御を要求する CSP 記述のパターンが現れている．同期イベントの意味を考慮し，これをつぎのように捉えることができる．

sync1->sync2 ： 金庫への紙幣取込完了待ち

sync1 ： 分岐区間への紙幣搬送完了待ち

この記述から，分岐区間を紙幣搬入区間および各金庫における共有資源として特定することができる．sync1，sync2 は分岐区間への操作を行なうコンポーネント間で排他制御を実現していることから，この同期イベントの順序に従い，分岐区間上で排他制御を行なうように再設計すると図 10 のようになる．

図 11 は，図 10 に基づいて共有資源を設計した結果である．プロセス Branch は，put を発生した後，pull5000 または pull1000 を繰り返すことが分かるので，これを実現する状態遷移を持つものとして設計している．

CSP を用いた共有資源上での排他制御を定義するための手順に従うことで，共有資源として分岐区間を特定し，その上での排他制御を行なうための順路式同等に CSP 記述を定義することができた．この CSP 記述に従って排他制御論理の実現が可能となる．

5. 関連研究

UML の状態機械図に対してモデル検査器を用いて検証を行なう試みは多く存在する．Ng ら [9] は，CSP を用いて UML の状態機械図を形式的に記述する方法を示し，モデル検査器 FDR を用いて検証できることを示している．Roscoe ら [10] は，状態遷移記述を CSP 記述に変換するコンパイラを定義し，FDR を用いてモデル検査を行なっている．Zhang ら [11] は，モデル検査器 PAT を定義している．状態遷移記述を CSP を拡張した CSP#による記述に変換

```
Entrance
    = put->Entrance
Branch
    = put->detect5000->pull5000->Branch
      [] put->detect1000->pull1000->Branch
FiveThousandBillStock
    = pull5000->FiveThousandBillStock
ThousandBillStock
    = pull1000->ThousandBillStock
BillSensor
    = detect5000->BillSensor
      [] detect1000->BillSensor

System
    = Branch
      [| msg |]
      ( Entrance
        ||| FiveThousandBillStock
        ||| ThousandBillStock
        ||| BillSensor )
msg = { put, pull5000, pull1000,
        detect5000, detect1000 }
```

図 10 共有資源に置き換えた振舞い

し，この記述を PAT の入力としている．Hsiung ら [12] は，組込みシステム向けの UML モデルからモデル検査やプログラミングを行なうためのアプリケーションフレームワークを提案している．これらは，対象としている図の種類および記述要素，モデルの検査内容が本研究とは異なる．

本研究では，CSP による振舞いのモデル化および共有資源上での排他制御を設計するための手順を提案した．提案手法では，CSP 記述と UML 記述，プログラムコードとの関係については示していない．関連研究 [9], [10], [11], [12] は，これらの関係を定義し，システムの振舞いの検証，設計の修正，実装を支援するものである一方で，複雑な並行プロセス間の協調について，適切な仕様の記述や修正を行なうための方法については示されていない．本研究の成果と，これらの成果を用いることにより，並行プロセス間の同期を定義し，その振舞いの検証を行なう一連のソフトウェアプロセスを支援することが可能となる．

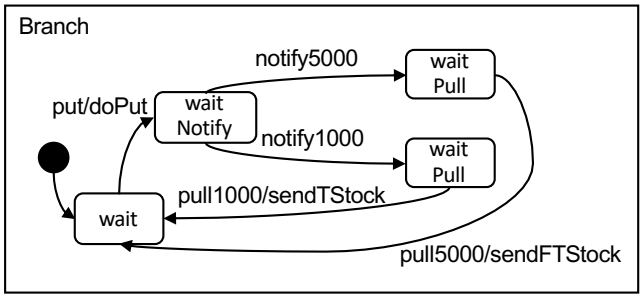

図 11 共有資源の設計

CSP 記述に基づいてコーディングするためのライブラリとして JCSP[13]，プログラミング言語として Go 言語 [14] がある．プロセスによってシステムを分割し，チャネルを用いたプロセス間通信を実現することができる．これらを用いることで CSP 記述を実現をすることが容易になる．一方，本研究では CSP 記述に基づいてオブジェクト指向による設計を行なうための手順を示した．

提案する設計法の基礎となるアーキテクチャでは，組込みシステムおよびそれを取り巻く環境の状態に対する構成や振舞いの変化論理をコンテキストアスペクトとして定義している．組込みシステムをはじめ，コンテキストへの自己適応機構を持つシステムの設計についても様々な試みが行なわれている．

Garlan ら [15] は，自己適応のためのモジュールを組み込んだ Rainbow アーキテクチャとそのフレームワークを提案している．自己適応計算の共通構造をアーキテクチャとして定義し，可変部をプラグインすることで，自己適応の実現を可能としている．Behjati ら [16] は，大規模組込みソフトウェアを対象としたアーキテクチャレベルの柔軟な構成変更方式を提案している．この研究では，組込みソフトウェアの構成変更をコンポーネント仕様間の制約充足問題ととらえ，構成と検証のためのアルゴリズムを提案している．Gerostathopoulos ら [17] は，組込みシステムにおける自己適応論理をベースレベルとメタレベルに分離させた構成を定義することで，多様なコンテキストへの柔軟な適応を可能としている．Chen ら [18] は，実行時の自己適応を実現するためのモデル変換機構を実現している．これらのアーキテクチャやフレームワーク，実行環境は，本研究と対立するものではなく，アーキテクチャ上のコンテキストアスペクトを設計，実現するための具体的な指針を与えるものと位置づけることができる．

複雑・多様化する組込みソフトウェアの設計においては，並行動作しながらコンテキストに応じて振舞いを変えるモジュール間の同期や排他制御を取り扱わなければならない．アスペクト干渉や優先度逆転といった好ましくない状況を避けるための振舞い設計法として，本研究で提案した共有資源の特定や排他制御設計方法は必ずしも十分とは言えない．コンテキストに応じた柔軟な並行制御設計方法として発展させるには，Wang ら [19] が提案しているオペレーティングシステムのための柔軟なスケジューリング方式や，Chen ら [20] が提案している自己適応システムのための多目的最適化方式などが参考になる．これらの検討については今後の課題としたい．

6. おわりに

本研究では，組込みソフトウェアのためのソフトウェアアーキテクチャに基づく設計法を定義した．ハードウェアデバイスの多様化とマイクロプロセッサの高性能化によ

り，組込みシステムの複雑化や高機能化が進んでいる．多種多様なハードウェアを制御し，多様な機能を実現する組込みソフトウェアには，システムの外部環境や内部状況に応じ，非機能要求を満たして動作することが求められる．

組込みシステムのためのアスペクト指向アーキテクチャに基づく設計プロセスとして，ハードウェアコンポーネントとそれに横断するアスペクトを独立して設計する方式を提案した．さらに，このソフトウェアアーキテクチャが規定する各構成要素の振舞い仕様を CSP によって記述することで，構成要素間の共有資源を特定し，排他制御論理を設計するためのプロセスを定義した．

提案する設計法を用いることで，構成要素の独立性を高めるとともに，並行・非同期処理において懸念されるデッドロック等の不具合の発生を抑制した組込みソフトウェア設計が可能となる．

紙幣搬送システムの例題を用いた事例検証では，ハードウェアコンポーネントとコンテキストアスペクトとの間のメッセージ通信に着目して, 共有資源の特定と排他制御論理の設計を行なえることを示した．

今後の課題としては，アーキテクチャに基づくフレームワークの構築，提案するプロセスを支援するための設計，検証ツールの整備などが挙げられる．

謝辞 本研究の一部は，科研費（基盤研究 (C) 19K11911, 20K11759）および 2021 年度南山大学パッヘ奨励金 I-A の助成による．

参考文献

[1] Walls, C.: *Embedded Software, Second Edition,* Newnes (2012).
[2] 江坂篤侍，野呂昌満，沢田篤史，繁田雅信，谷口弘一：コンテキストアウェアネスを考慮した組込みシステムのためのアスペクト指向アーキテクチャの設計, ソフトウェア工学の基礎 XXIV（日本ソフトウェア科学会 FOSE2017），pp.3–12 (2017).
[3] 江坂篤侍，野呂昌満，繁田雅信，沢田篤史：ソフトウェアアーキテクチャに基づく組込みシステムの設計法に関する研究，ソフトウェア工学の基礎 XXVI（日本ソフトウェア科学会 FOSE2019），pp.151–156 (2019).
[4] Hirschfeld, R., Costanza, P. and Nierstrasz, O.: Context-oriented Programming, *Journal of Object Technology,* Vol.7, No.3, pp.125–151 (2008).
[5] Bass, L., Clements, P. and Kazman, R.: *Software Architecture in Practice, Third Edition,* Addison-Wesley (2012).
[6] 江坂篤侍，野呂昌満，沢田篤史：インタラクティブシステムのための共通アーキテクチャの設計，コンピュータソフトウェア，Vol.35, No.4, pp.3–15 (2018).
[7] Campbell, R.H. and Habermann, A.N.: The Specification of Process Synchronization by Path Expressions, *Operating Systems,* Gelenbe, E. and Kaiser, C. (eds.), Springer, pp.89–102 (1974).
[8] Gibson-Robinson, T., Armstrong, P., Boulgakov, A. and Roscoe, A.W.: FDR3 – a Modern Refinement Checker for CSP, *Proc. International Conference on Tools and Algorithms for the Construction and Analysis of Systems,* Springer, pp.187–201 (2014).
[9] Ng, M. Y. and Butler, M.: Towards formalizing UML State Diagrams in CSP, *Proc. First International Conference on Software Engineering and Formal Methods,* IEEE, pp.138–147 (2003).
[10] Roscoe, A. W. and Wu, Z.: Verifying Statemate Statecharts using CSP and FDR, *Proc. International Conference on Formal Engineering Methods,* Springer, pp.324–341 (2006).
[11] Zhang, S.J. and Liu, Y.: An Automatic Approach to Model Checking UML State Machines, *Proc. 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion,* IEEE, pp.1–6 (2010).
[12] Hsiung, P.A., Lin, S.W., Tseng, C.H., Lee, T.Y., Fu, J.M. and See, W.B.: VERTAF: An Application Framework for the Design and Verification of Embedded Real-time Software, *IEEE Transactions on Software Engineering,* Vol.30, No.10 (2004).
[13] Welch, P.: Communicating Sequential Process for Java (JCSP) (online), available from ⟨https://www.cs.kent.ac.uk/projects/ofa/jcsp/⟩ (2021.08.02).
[14] Donovan, A.A. and Kernighan, B.W.: *The Go Programming Language,* Addison-Wesley Professional (2015).
[15] Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B. and Steenkiste, P.: Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure, *Computer,* Vol.53, No.10, pp.46–54 (2004).
[16] Behjati, R., Nejati, S. and Briand, L.C.: Architecture-Level Configuration of Large-Scale Embedded Software Systems, *ACM Transactions on Software Engineering and Methodology,* Vol.23, No.3, Article 25 (2014).
[17] Gerostathopoulos, I., Bures T. Hnetynka, P., Hujecek, A., Plasil, F. and Skoda, D.: Strengthening Adaptation in Cyber-Physical Systems via Meta-Adaptation Strategies, *ACM Transactions on Cyber-Physical Systems,* Vol.1, No.3, Article 13 (2017).
[18] Chen, B., Peng, X., Yu, Y., Nuseibeh, B. and Zhao, W.: Self-Adaptation through Incremental Generative Model Transformation at Runtime, *Proc. International Conference on Software Engineering,* pp.676–687 (2014).
[19] Wang, H., Xu, C., Guo, B., Ma, X. and Lu, J.: Generic Adaptive Scheduling for Efficient Context Inconsistency Detection, *IEEE Transactions on Software Engineering,* Vol.47, No.3, pp.464–497 (2021).
[20] Chen, T., Li, K., Bahsoon, R. and Yao, X.: FEMOSAA: Feature-Guided and Knee-Driven Multi-Objective Optimization for Self-Adaptive Software, *ACM Transactions on Software Engineering and Methodology,* Vol.27, No.2, Article 5 (2018).