Vol.2016-SE-193 No.11
2016/7/13

組込みシステムへのコンテキスト指向プログラミング技術の適用

江坂 篤侍[1]　野呂 昌満[2]　沢田 篤史[2]　繁田 雅信[3]　谷口 弘一[3]

概要：一般に組み込みシステムにおいては，センサの検知する値の組み合わせに応じたアクチュエータ群の協調動作を記述することは難しい．これはセンサの検知する値が同じでも，システムの一部または全体の状態によってアクチュエータ群が異なる動作をすることに起因する．実際に長年利用されてきた組込みシステムのプログラムでは，何度も改版を受けた結果，システムの状態に応じた動作を実現するために 1 つの条件式が何千行にも渡って記述され，またそれが幾つも散在するようなこともある．本研究の目的は，このような組込みシステムを整理するために，コンテキストアウェアネスプログラミング技術を適用することである．一方，組込みシステムにおいては実時間性，耐故障性，並行性などの非機能特性が重視され，これらはアスペクト指向計算においては横断的関心事とされる．我々は，コンテキストおよびその他の横断的関心事をアスペクトとして分離したアスペクト指向アプリケーションアーキテクチャを設計した．紙幣搬送システムを事例としてアスペクト指向アーキテクチャによる実現の妥当性を確認した．

Describing Embedded System's Structures using Context Oriented Programming

Esaka Atsushi[1]　Noro Masami[2]　Sawada Atsushi[2]　Shigeta Masanobu[3]　Taniguchi Kouichi[3]

Abstract: There is a class of embedded systems which have very complicated control structures in order to handle the systems' complex behaviors. In such system, comprising actuators behave differently according to system's internal status. Programming the control logic of these actuators consistently is often very difficult. In this paper we apply the notion of context aware programming to the class of embedded systems to make their structures easy to understand and maintain. On the other hand, non-functional properties including real-time, fault tolerant, concurrency and so forth are commonly important for embedded systems. These non-functional properties are often recognized as cross-cutting concerns over the system's dominant (i.e. object oriented or procedural) decomposition. We have recognized the context-awareness as an additional crosscutting concern and developed an aspect-oriented software architecture for embedded systems. We have demonstrated the validity of our architecture using a case study of the bill conveyer system development.

1. はじめに

組み込みシステムのプログラムを簡略化し保守が容易になるように整理するためには，システムの状態の制御処理，センサの処理，アクチュエータの処理およびこれらの並行動作による協調の処理をそれぞれモジュールとして独立に記述し，これを管理できるようにすることが重要である．

本研究の目的は，以上に述べた特徴を持つ組込みシステムを整理するために，次のように組み込みシステムのソフトウェアアーキテクチャを定義することである．

1. オブジェクト指向によるモジュール構造を基礎とする
2. コンテキスト指向によって制御構造の条件と処理を分離する
3. アスペクト指向によって横断的関心事を分離する

組込みシステムは物理的な対象を制御するものであること

[1] 南山大学大学院数理情報研究科
Graduate School of Mathematical Sciences and Information Engineering, Nanzan University
[2] 南山大学理工学部ソフトウェア工学科
Department of Software Engineering, Nanzan University
[3] 富士電機株式会社
Fuji Electric Co., Ltd.

からオブジェクト指向に基づくモジュール分割が向いている．並行に動作するオブジェクトは，イベントの授受によって協調することから状態遷移機械によって実現することが自然である．システムの状態をコンテキストとし，コンテキストの変化に応じたアクチュエータ群の動きをレイヤとすれば，コンテキストアウェアネスプログラミング技術を適用できる．コンテキストアウェアネスプログラミング技術により，制御構造における条件と処理を分離することで，その複雑さを軽減する．通常組込みシステムに重視すべき非機能特性である並行性，実時間性，耐故障性などは横断的関心事となる．これら横断的関心事を分離しアスペクトの集合とするアスペクト指向アーキテクチャとして定義することで，実装や保守を容易にする．

信頼性の保証を目的とし，次の手順によって並行に動作する状態遷移機械間の同期を実現する．

1) センサ，アクチュエータの挙動を CSP [3] で記述
2) 同期のためのイベントに着目し，共有資源を定義
3) 共有資源上での排他制御を定義

紙幣等紙状のものを搬送・管理するシステム (以下，紙幣搬送システム) を事例として，我々の提案するアスペクト指向アーキテクチャによる実現の妥当性を確認した．紙幣搬送システムは，搬送対象の保管状態および搬送状態に応じて複数のアクチュエータの動作が変わることから，挙動が複雑な組込みシステムの事例として適していると考えた．

2. 組込みシステムのためのアスペクト指向コンテキストアウェアアーキテクチャの設計

2.1 組込みシステムとコンテキスト指向プログラミング

組み込みシステムは，システム外部からのボタン押下やレバーを引くなどの刺激 (外部イベント) をセンサが検知し，アクチュエータに通知することによってシステム全体としての動作を実現する．組込み機器を制御するソフトウェアは，並行性と外部イベント処理を実現しなければならない．

多くの組込みソフトウェアは，手続き指向に基づいて実現されている．ハードウェアの動作を安直に実現するならば，外部イベントによる割込みに対応する特定のハードウェアへの操作を１つの手続きにして実現する．ハードウェアの振舞いは，イベントの種類と順序および組込みシステム内部の状態によって変わるので，この手続きはこれらの条件を複合的に実現する．この条件を実現するプログラムでは，if 文や case 文が何重もの入れ子になる場合が多い．複雑な条件とそれに対する処理が一体となって記述され，それが何千行にもおよぶこともあるので保守が困難となる．

コンテキスト指向プログラミングは，実行時に参照される情報をコンテキストとし，コンテキストの変化に応じた処理を実行する [2]．このコンテキストの変化に応じて適用される処理群をレイヤとしてグループ化する．組込みシステムに対してコンテキストアウェアネスプログラミング技術を適用した場合，特定のセンサ群の検知し得る値と機器の状態の集合をコンテキスト，コンテキストの変化に応じてアクチュエータを起動する処理をレイヤとしてモジュール化する．手続き指向に基づく実現の制御構造と比較して，条件とそれに対する処理を分離するので，複雑さは軽減され，実装や保守が容易になる．

2.2 アスペクト指向コンテキストアウェアネスアーキテクチャの設計

本研究では，アスペクト指向技術 [7] を適用し，組込みソフトウェアのためのアスペクト指向アーキテクチャを設計した．設計したアーキテクチャを図 1 に示す．組み込みソフトウェアが持つ複数の横断的関心事をアスペクト指向技術を適用してそれぞれ分離してモジュール化した．オブジェクト指向コンサーンおよびコンテキスト指向コンサーンをプライマリーコンサーンとし，並行性，実時間性，耐故障性コンサーンをセカンダリーコンサーンとして設計した．

ソフトウェア記述において計算モデルに関する関心事が基本的なモジュール構造を定義することから，オブジェクト指向コンサーンをプライマリーコンサーンとすることが自然であると考えた．図 1 中央部の丸実線枠の Hardware，Sensor，Actuator，SensorActuator はオブジェクト指向によって定義されるモジュール構造を示す．ハードウェアの多相型としてセンサ，アクチュエータ，センサアクチュエータとして定義することにより，これらを総称的に扱うことを可能とした．オブジェクト指向はハードウェア単位でモジュール化することから，ハードウェアの構成の変更に対してソフトウェアが柔軟に対応可能である．

上述の通り，コンテキスト指向によって，制御構造が整理されることから，オブジェクト指向によって定義されるモジュール間の協調に対し，コンテキスト指向を適用する．Shaw ら [6] は，イベント処理を行なうシステムのためのアーキテクチャスタイルとしてイベントベーススタイルを提案している．今日では，これを具体化したアーキテクチャスタイルとしてコンテキスト指向スタイル [2] が提案されていることから，我々はこれを導入した．図 1 中央部の丸破線枠は，オブジェクト指向に基づくコンテキスト指向コンサーンによるモジュール構造を示す．イベントの種類と順序および機器の状態の集合をコンテキスト，コンテキストの変化に応じてアクチュエータを起動する処理をレイヤとして定義した．また，イベント通知による協調を前提としたオブジェクトは，状態遷移機械として実現することが自然と考えた．このようにオブジェクト指向とコンテキスト指向に基づくモジュール分割を組み合わせることにより，条件と条件に応じたハードウェア間の協調処理がそ

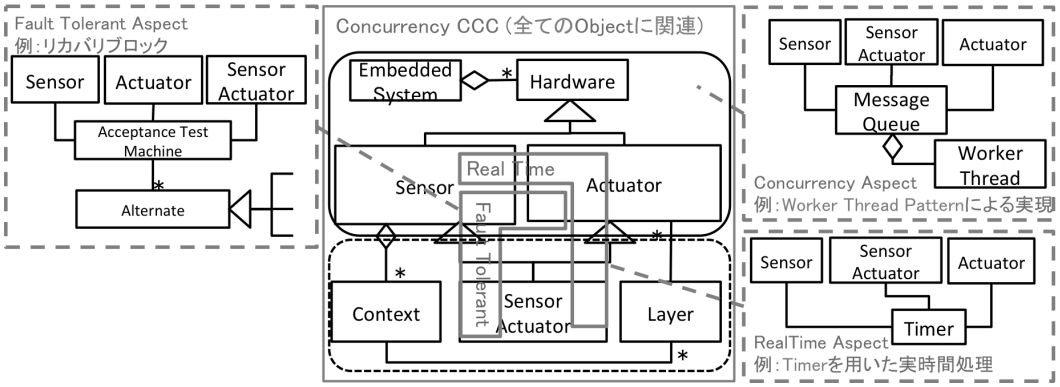

図 1 設計したアスペクト指向アーキテクチャ

れぞれモジュール化されることから，制御構造の複雑さは軽減され，実装や保守が容易になる．

過去の開発経験 [5] に基づく知見から，組込みソフトウェアにおいて並行性，実時間性，耐故障性が重視されることから，これらをセカンダリーコンサーンとした．図 1 灰色枠には，プライマリーコンサーンによって定義されるモジュール構造とこれらセカンダリーコンサーンとの関係を示している．並行性コンサーンは，複数のハードウェア間に並行動作を実現するための処理が横断することを示している．また，実時間性コンサーンおよび耐故障性コンサーンは，センサおよびアクチュエータのアクションにこの特性に関する処理が横断することを示している．図 1 破線枠部分には，分離された並行性，実時間性，耐故障性コンサーンによって定義されるモジュール構造の例を示している．例えば，図 1 では耐故障性コンサーンによって定義されるモジュール構造はリカバリブロックを実現する構造を示している．

3. アスペクト指向コンテキストアウェアネスアーキテクチャに基づく組込みシステムの開発

本章では，我々の提案したアーキテクチャに基づく開発について説明する．それぞれの関心事に適した構造についてデザインパターンに着目し，実現アーキテクチャを定義した．以降より，実現アーキテクチャおよびこれに基づく開発プロセスについて説明する．この開発における省力化および信頼性の保証の方法について説明する．

3.1 実現アーキテクチャ

コンテキスト指向およびアスペクト指向を導入するさいに，コンテキスト指向プログラミング言語およびアスペクト指向プログラミング言語を用いることは自然である．しかし，特定の言語に依存することから，汎用性を考慮してこれらを利用しないこととした．代わりに，コンテキスト指向およびアスペクト指向の実現にはデザインパターン [1] を用いて定義した．以下にそれぞの関心事について適用したデザインパターンを説明する．

オブジェクトの実現には，ステートパターンおよびコマンドパターンを用いる．オブジェクトは状態遷移機械として実現することから，状態の実現にステートパターン，アクションの実現にコマンドパターンを用いる．

コンテキストに応じたハードウェアの動作の変化は，ハードウェアの多相型を定義することで実現する．オブジェクト指向では，振舞いのバリエーションを表現するために多相型を定義する．総称的に取り扱うことが可能となることから，コンテキストの変化に応じた起動するアクチュエータの変化は，インスタンスの入替えによって実現する．

レイヤの実現には，ファクトリーメソッドパターンを用いる．レイヤが活性化されたさいに，適切なアクチュエータのインスタンスを生成し，イベントを通知する．このインスタンスを生成する手続きを生成パターンであるファクトリーメソッドパターンを用いてモジュール化する．

コンテキストの実現には，論理型オブジェクトとコマンドパターンを用いて実現する．コンテキストは，現在の状態に応じて特定のレイヤを活性化する．現在の状態を判定する論理式の実現には，SmallTalk でも定義されている論理型オブジェクトが必須となる．論理型オブジェクトにより現在のセンサの値を評価し，この結果に応じてコマンドによってレイヤを活性化させる．

並行性コンサーンによって定義される並行処理は，ワーカースレッドパターンを用いて実現する．センサとアクチュエータ間のイベント通知に割り込み，モニタ等を用いて実現される．

実時間性コンサーンによって定義される実時間処理は，タイマーオブジェクトを定義することで実現する．タイマーによるタイムアウトイベント通知によって，実時間処理を実現する．

耐故障性コンサーンによって定義される耐故障処理は，リカバリブロックや N バージョンプログラミング，N セルフチェッキングプログラミングによって実現する．

3.2 開発プロセス

アスペクト指向アーキテクチャに基づく開発プロセスは，次のように定義できる．

1. プライマリーコンサーンによるアーキテクチャに基づく設計，コーディング
2. セカンダリーコンサーンによるアーキテクチャに基づく設計，コーディング
3. アスペクト間記述を実現

アーキテクチャを定義することは，同時にそのモジュール群を開発するためのプロセスを定義することになる [4]．アスペクト指向アーキテクチャに基づく開発においては，横断的関心事によって定義されるモジュール構造は独立して定義されることから，それぞれを並行して開発を行なうことが可能である．

図 1 のアスペクト指向アーキテクチャから具体的な組込みソフトウェアの開発プロセスが定義できる．図 2 に，この開発プロセスを示す．図 2 灰色枠は，それぞれの関心事に関連する開発プロセスを示している．セカンダリーコンサーンに関連する開発を行なうには，それが横断するモジュール群が定義されている必要がある．並行性コンサーンはオブジェクト間の並行処理を実現することから，オブジェクト間の協調が定義された後に開発が行なわれる．また，実時間性，耐故障性コンサーンは状態遷移機械のアクションに横断することから状態遷移機械のアクションの論理が設計された後に開発が行なわれる．オブジェクトの振舞いの定義後に，その構造を洗練することが多いので，スパイラル型開発プロセスモデルを前提としている．

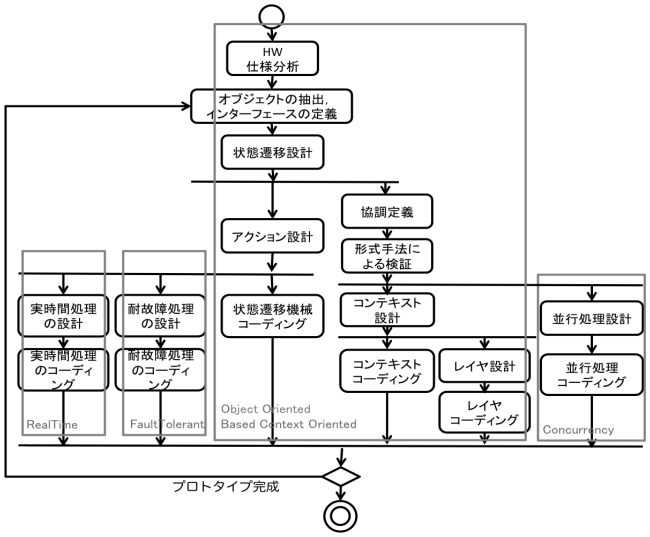

図 2 設計したアーキテクチャに基づく開発プロセス

一般にアプリケーション開発の省力化を目的として，アーキテクチャの実現としてアプリケーションフレームワークが定義される．ホットスポットのカスタマイズコードの自動生成は一般に困難である．しかし，アプリケーションドメインを限定することにより仕様とカスタマイズコードの間にパターンが現れる．これにより，一部のホットスポットのカスタマイズコードについては，生成またはサブアプリケーションフレームワークを生成可能である．我々の設計したアーキテクチャに基づくアプリケーションフレームワークを定義し，さらに仕様とコードにパターンが存在するものについてカスタマイズコードの自動生成を実現する．

アプリケーションの信頼性の保証を目的として，並行に動作するハードウェアの振舞いを形式的に記述し，系統的な手法により検証する．図 2 における協調定義を行なうさいには，次の手順により並行に動作するハードウェア間の同期を実現する．

1. CSP によるハードウェアの振舞いのモデル化
2. 同期イベントに着目して共有資源を特定
3. 順路式を定義し，共有資源上での排他制御の実現

並行に動作する状態遷移機械の振舞いを形式的に記述するために，プロセス代数理論として代表的な CSP を用いる．CSP 記述における同期イベントは，特定のハードウェアを介した協調を実現する場合があることから，その同期イベントの意味を考慮することで，共有資源が特定できる．共有資源への操作の実行順序を制限する順路式を定義することでハードウェア間の同期を実現する [8]．CSP 記述におけるこの共有資源の振舞いの系列は，このハードウェアに対する操作の順序を定義することから，この振舞いの系列に基づいて順路式が定義できる．

3.3 アーキテクチャに基づく開発の省力化

3.1 の実現アーキテクチャに基づき，アプリケーションフレームワークを定義する．センサ，アクチュエータ，コンテキスト，レイヤはアプリケーション毎に具体化して実現されることから，ホットスポットとなる．アプリケーションの仕様と実現されるこのホットスポットのカスタマイズコードにパターンが存在し，一部については自動生成可能であることから，サブアプリケーションフレームワークを生成できる．

センサ，アクチュエータを実現する状態遷移機械において，イベント，状態，アクションそれぞれの組み合わせはアプリケーションによって異なる．状態オブジェクトの振舞いは，イベントに応じてアクションを起動し，次の状態に遷移することから，指定した組み合わせから全て生成可能である．アクションオブジェクトは，その振舞いの実装にアプリケーション特有の論理を含むことから，一部手書きによる実装が必要である．以上のことから，サブアプリケーションフレームワークを生成可能である．

コンテキストにおいて，論理型オブジェクトとレイヤを活性化させるコマンドはアプリケーションによって異なる．論理型オブジェクトに定義される評価の処理や，コマンドオブジェクトに定義されるレイヤの活性化処理はアプ

リケーション特有の記述となる．したがって，論理型オブジェクトおよびコマンドオブジェクトそれぞれについて，手書きによってこの処理を実装する必要がある．論理型オブジェクトの組み合わせと，評価の結果に応じて活性化させるコマンドを指定することで，コンテキストを生成可能である．以上のことから，サブアプリケーションフレームワークとして定義できる．

レイヤにおいて，イベントを通知する対象のハードウェアはアプリケーションによって異なる．このハードウェアの多相型はアプリケーション特有なので，ファクトリーメソッドの論理を手書きで実装する必要がある．対象の型のインスタンスを生成するファクトリーメソッドを指定することで，レイヤを生成可能である．以上のことから，サブアプリケーションフレームワークを生成可能である．

3.4 信頼性の保証

それぞれのハードウェアは複雑に協調することから，これを把握することは困難であり，誤った設計によってデッドロックなどの問題が起こり得る．ハードウェアは並行に動作し，イベント授受によって協調することから，我々はその振舞いを CSP モデルに基づいて記述し，これを検証する．例えば，3 つのオブジェクト (Obj1，Obj2，Obj3) によって構成されるシステムの挙動を CSP モデルに基づいて記述すると CSP 記述 1 になるとする．それぞれのオブジェクトは，CSP 記述におけるイベント (Ev1，Ev2，Ev3，Ev4，Ev5) に対応するアクションを実行する．Sync1，Sync2，Sync3 は，このオブジェクト間の同期イベントである．

CSP 記述 1：Obj1，Obj2，Obj3 の振舞い

```
1 Obj1=Obj1.Ev1–>Sync1–>Sync3–>Obj1.Ev5 –> Obj1
2 Obj2=Sync2–> Obj2.Ev3 –> Sync3 –> Sync1 –> Obj2
3 Obj3=Sync1–>Obj3.Ev2–>Sync2–>Sync3
4           –>Obj3.Ev4 –> Obj3
```

CSP 記述における同期イベントに着目して，共通資源を特定する．連続して同期イベントが現れた場合，特定の共有資源を介した協調を表す場合がある．同期イベントに示されるように Obj3 を介して Obj1 と Obj2 は協調していることから，Obj3 は Obj1 と Obj2 の共有資源である．同期イベント発生時のイベント授受を表現するために，CSP 記述を拡張し，同期イベントを {Act:...} に置き換えた．この記述はアクションによって他のオブジェクトにイベントを通知することを示す．結果として，CSP 記述 1 から共有資源 Obj3 を特定し，CSP 記述 2 では共有資源 Obj3 を介したハードウェア間の協調を示すことができた．

CSP 記述 2：Obj1，Obj2，Obj3 のイベント授受

```
1 Obj1=Obj1.Ev1{Act: Obj3.Ev2} –>Obj1.Ev5 –> Obj1
2 Obj2=Obj2.Ev3{Act: Obj3.Ev4} –> Obj2
3 Obj3=Obj3.Ev2{Act: Obj2.Ev3}
4           –>Obj3.Ev4{Act:Obj1.Ev5} –> Obj3
```

特定した共有資源への操作の実行順序を制限する順路式を定義することでハードウェア間の同期を実現する．CSP 記述 2 に示すように，共有資源 Obj3 は Ev2 に対応するアクションの実行の後に，Ev4 に対応するアクションを実行する．したがって共有資源 Obj3 の順路式は **path** (Ev2;Ev4)* **end** となる．これにより，Obj1 と Obj2 間の同期が実現される．

4. 事例による考察

紙幣等紙状のものを搬送・管理するシステム (以下，紙幣搬送システム) を事例として，設計したアスペクト指向アーキテクチャによる実現の妥当性を考察する．紙幣搬送システムにおいて，センサの検知する搬送対象の保管状態および搬送状態に応じて複数のアクチュエータの動作が変わることからコンテキスト指向を適用することは適切である．

3.2 の開発プロセスに基づいて紙幣搬送システムの設計を行なう．協調を定義するさいには，信頼性の保証を目的として，3.4 に述べたように，紙幣搬送システムの振舞いをモデル化し，このモデルに基づいて協調を設計する．

4.1 対象とする機器の概要

紙幣搬送システムのハードウェアの構成を図 3 に示す．紙幣搬送システムには，搬送路と保管庫がある．搬送路上には，搬送対象を搬送するための搬送路モータと搬送対象の有無を検知する搬送路センサがある．保管庫には，スタッカ，ストッカがある．ストッカは搬送対象を保管する場所であり，スタッカは搬送対象を一時的に保管する場所である．スタッカ，ストッカそれぞれには搬送対象の有無を検知するスタックセンサと紙幣ストックセンサがある．搬送路と保管庫との間にはゲートがある．

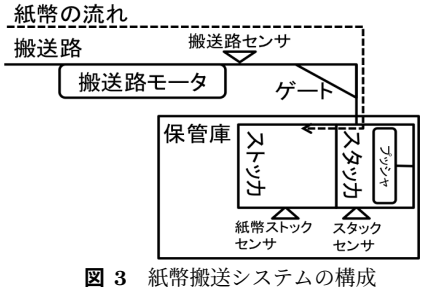

図 3 紙幣搬送システムの構成

紙幣搬送システムは，搬送路上の搬送対象をストッカに格納する．搬送路センサが搬送対象を検知し，スタックセンサが搬送対象を検知していない時，搬送路モータは正方向に回転し，ゲートは開放される．ゲートが開放時，搬送対象を搬送路からスタッカに格納可能となる．スタックセンサが搬送対象を検知し，紙幣ストックセンサが搬送対象

を検知していない時，待機位置のプッシャが搬送対象をストッカに押し込む．紙幣ストックセンサが搬送対象を検知したら，プッシャは待機位置に移動する．搬送路センサが搬送対象を検知しなくなったら，搬送路モータは停止し，ゲートは閉鎖される．

4.2 紙幣搬送システムの設計

コンテキストアウェアネスプログラミング技術を適用しなかった場合，アクチュエータには，複数のセンサの検知する値の組み合わせによる場合分けが記述される．例えば，プッシャは，紙幣ストックセンサの値，スタックセンサの値，ゲートの状態，搬送路センサの状態のそれぞれの組み合わせに応じて待機位置またはストッカ位置に移動する．紙幣ストッカの種類が増えた場合，プッシャはそれぞれの紙幣ストッカに振り分けることから，その組み合わせはさらに複雑になる．したがって，組込みシステムへコンテキストアウェアネスプログラミング技術の適用を説明するための事例として十分である．

図 4 は，紙幣搬送システムのアプリケーションアーキテクチャである．図 3 から特定したハードウェアをセンサとアクチュエータに分類し，多相型として表現している．また，前述の紙幣搬送システムの動きからセンサの値の集合をコンテキスト，アクチュエータの集合をレイヤとして定義した．前述の紙幣搬送システムの振舞いの説明から，Transport Sensor が搬送対象を検知し，Stack Sensor が搬送対象を検知しない時，搬送路上の搬送対象はスタッカに搬送されることがわかる．したがって，Transport Context は，Transport Sensor と Stack Sensor の値の組み合わせをスタッカへの搬送時のコンテキストとして定義した．この時に搬送を実現するアクチュエータの集合のレイヤを TransportLayer として定義した．

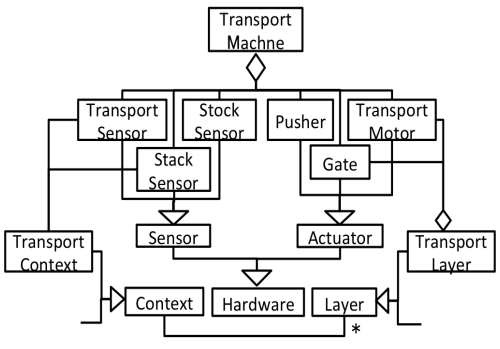

図 4 紙幣搬送システムのアーキテクチャ

それぞれのハードウェアのインターフェースを表 1 のように設計した．これは，前述の紙幣搬送システムの動きから，それぞれのハードウェア毎にその動作を抽出して定義した．例えば，搬送路モータは，停止と正方向回転を行なうことから，インターフェースとして，停止を行なう nd_stop，正方向回転を行なう nd_start を定義している．ハードウェアの振舞いの順序関係を整理し，状態遷移機械として図 5 のように設計した．前述の紙幣搬送システムの動きから，搬送路モータは，停止，正方向回転を繰り返すことがわかる．したがって，初期状態は停止であり，その後，正方向回転を行なう状態遷移機械として定義している．

表 1 紙幣搬送システムにおける各 Hardware のインターフェース

Hardware	インターフェース	説明
搬送路/スタック/紙幣ストックセンサ	yes	搬送対象を検知
	no	搬送対象を検知しない
搬送路モータ	nd_start	正方向回転
	nd_stop	停止
ゲート	open	ゲート開放
	close	ゲートを閉鎖
プッシャ	twp	待機位置に移動
	tsp	ストッカ位置に移動

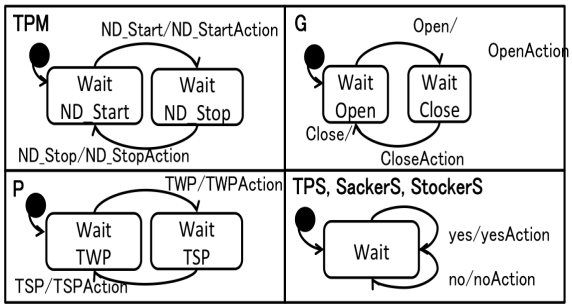

図 5 センサ，アクチュエータの設計

4.3 信頼性の保証

3.2 に述べた手順に従って，それぞれのハードウェアの振舞いを CSP モデルに基づいてに記述し，共有資源を定義，共有資源上での排他制御の実現を行なう．

4.3.1 CSP によるハードウェアの振舞いのモデル化

前述のハードウェアそれぞれの設計に基づいて，その振舞いを CSP モデルに基づいて記述する．結果として記述された CSP 記述を CSP 記述 3 に示す．この記述では，同期イベント (Sync1，Sync2，Sync3，Sync4) により，ハードウェア間の協調を実現している．この CSP 記述における同期イベントに着目し，共有資源を特定する．

CSP 記述 3：紙幣搬送システムの構成要素の振舞い

```
1 TPM = Sync1 -> TPM.nd_start -> Sync2 -> Sync3
2               -> TPM.nd_stop -> TPM
3 G   = Sync1 -> G.open -> Sync2 -> Sync3
4               -> G.close -> G
5 P   = P.twp -> Sync3 -> P.tsp
6               -> Sync4 -> Sync1 -> P
```

```
StackS  = Sync2 -> StackS,yes -> Sync3
                -> Sync4 -> StackS.no -> StackS
StockS  = Sync4 -> StockS.yes -> StockS
                | StockS.no -> StockS
TPS = TPS.yes -> Sync1 -> TPS.no -> TPS
Process = (((((TPM [| Sync1 - 4 |] G )
                   [| Sync1 - 4 |] P )
                   [| Sync1 - 4 |] StackS )
                   [| Sync1 - 4 |] StockS )
                   [| Sync1 - 4 |] TPS )
(搬送路モータ = TMP(TransportMotor)
ゲート = G(Gate), プッシャ = P(Pusher)
搬送路センサ = TPS(Transport Sensor)
スタックセンサ = StackS(Stacker Sensor)
紙幣ストックセンサ = StockS(Stock Sensor))
```

4.3.2 同期イベントに着目した共有資源の特定

それぞれのアクチュエータには連続して同期イベントが現れている．それぞれの振舞いに定義されるイベントの意味を考慮すれば，同期イベントは具体的に次のように捉えることができる．

Sync2-3 ：スタッカに搬送対象投入完了待ち
Sync3-4：ストッカに搬送対象投入完了待ち
Sync4-1 ：搬送路に搬送対象配置完了待ち

アクチュエータがセンサの検知する対象へ操作を行なった結果，センサの値が変化し，この値の変化をきっかけとして，アクチュエータ間で協調していると考えることができる．センサが監視する対象の搬送路，スタッカ，ストッカを介してアクチュエータ間の協調が実現されることから，この監視する対象は共有資源である．CSP 記述 3 における TPS，StackerS，StockerS を，監視対象である搬送路，スタッカ，ストッカ (TP，Stacker，Stocker) に置き換え，同期イベントを共有資源とのイベントの授受に置き換えると CSP 記述 4 のようになる．共有資源である搬送路，スタッカ，ストッカのインターフェースとして put，get を定義した．各センサの yes イベントは，対象で搬送対象が検知されたことを示すことから，共有資源に対する搬送対象の格納 (put) イベントに置き換えた．各センサの no イベントは，対象で搬送対象が検知されていないことを示すことから，共有資源に対する搬送対象の取得 (get) イベントに置き換えた．同期イベントを {Act:...} に置き換えることでハードウェア間のイベント授受を表現した．結果として，CSP 記述 4 では，共有資源として搬送路，スタッカ，ストッカを特定し，共有資源を介したハードウェア間の協調を示すことができた．

CSP 記述 4：共有資源に置き換えた振舞い

```
TPM = TPM.nd_start -> TPM.nd_stop -> TPM
G   = G.open -> G.close -> G
P   = P.twp -> P.tsp -> P
Stacker  = Stacker.put
           {Act: TPM.nd_stop, G.close,P.tsp}
           -> Stacker.get{Act: P.twp}-> Stacker
Stocker  = Stocker.put -> Stocker
         | Stocker.get -> Stocker
TP = TP.put{Act: TPM.nd_start, G.open}
           -> TP.get -> TP
(搬送路 = TP(Transport)
スタッカ = Stacker, ストッカ = Stocker)
```

4.3.3 順路式を定義することによる共有資源上での排他制御の実現

センサおよびアクチュエータの間の協調の設計結果 (CSP 記述 4) に基づいて，共有資源における順路式を定義し，共有資源を設計する．この記述では共有資源に対する操作の順序関係を定義していることから，この振舞いの系列がそれぞれ共有資源の順路式となる．例えば，共有資源 Stacker は，put 操作の後に get 操作が行われることから，この順路式は **path** (put;get)* **end** となる．図 6 は，この順路式に基づいて共有資源を設計した結果である．例えば，Stacker は put，get の繰り返しであることから，初期状態を WaitPut とし，put イベントを受理したら WaitGet 状態に遷移，さらに get イベントを受理したら WaitPut 状態に遷移する状態遷移機械として設計している．実際に紙

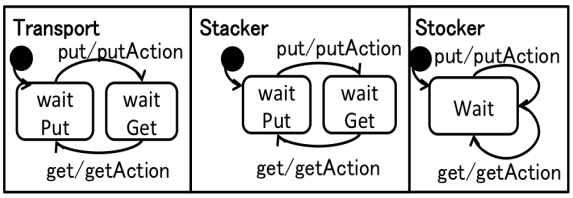

図 6 共有資源の設計

幣搬送システムにおいて，共有資源を特定し，その上での排他制御を行なうための順路式が定義できた．形式的に記述された振舞いについて検査を行なうことで信頼性を保証することが可能となる．

5. アプリケーションフレームワーク導入の可能性に関する考察

アプリケーションフレームワークの導入の可能性を考察する．3.3 で述べた仕様に対応するコードのパターンに基づき，紙幣搬送システムを例として具体的な仕様と出力されるコードが定義できることから，ホットスポットのカスタマイズコードの自動生成を実現可能であると考える．

状態遷移機械は，状態とその状態で受理可能なイベントおよび起動されるアクションの関係に基づきそのコードを定義することが可能である．例えば，図 5 に示した状態遷移図はこの関係を示している．ゲートの状態遷移図に記述される状態から，WaitOpen，WaitClose の状態クラスが生成される．状態遷移のアクション部分から，OpenAction，

CloseAction のアクションクラスが生成される．WaitOpen の振舞いは，状態と状態遷移に記述される内容から，Open イベントを受理したら，OpenAction を起動するものとして生成される．OpenAction の振舞いはゲート特有なものなので手書きによる実装が必要である．

コンテキストは，センサの値の組み合わせと活性化されるレイヤの関係に基づきそのコードを定義することが可能である．表 2 は，例として Transport Context におけるこの関係を示している．センサ列と検知した値列から，それぞれのセンサ毎の論理型オブジェクトを生成する．活性化するレイヤ列から，このレイヤを活性化するコマンドオブジェクトを生成する．生成された論理型オブジェクトによりセンサの値を評価し，結果が全て正だった場合に，活性化するレイヤ列のレイヤに活性化イベントが通知される．論理型オブジェクトおよびコマンドオブジェクトに対して手書きによってそれぞれの処理をコーディングする必要がある．

表 2 Transport Context の設計

センサ	検知した値	活性化するレイヤ
搬送路センサ	搬送対象有	Transport
スタックセンサ	搬送対象無	Layer

レイヤは，アクチュエータと通知するイベントの関係に基づきそのコードを定義することが可能である．表 3 は，例として Transport Layer におけるこの関係を示している．このレイヤが活性化したら，搬送路モータに正回転開始 (nd_start) イベントを通知し，ゲートに開放 (open) イベントを通知する．それぞれのアクチュエータのインスタンスを生成するための手続きは，ファクトリーメソッドパターンとして実現されるが，この手続きはアクチュエータ毎に異なるので手書きが必要である．

表 3 Transport Layer の設計

アクチュエータ	通知するイベント
搬送路モータ	正回転
ゲート	開放

3.3 では異なるアプリケーションで共通する部分を明らかにし，アプリケーションフレームワークを定義できることを述べた．紙幣搬送システムについて，仕様に対応するコードのパターンに従って，具体的な仕様とコードが定義できることを確認できたことから，ホットスポットのカスタマイズコードの自動生成を実現可能である．

6. おわりに

本稿では，組込みソフトウェアへのコンテキストアウェアネスプログラミング技術の適用について考察した．組込みソフトウェアには，複数の横断的関心事が存在し，それぞれ異なるモジュール分割の視点を与える．このことから，組込みソフトウェアのためのアスペクト指向アーキテクチャを設計した．プライマリーコンサーンをオブジェクト指向に基づくコンテキスト指向コンサーンとし，セカンダリーコンサーンを並行性，実時間性，耐故障性コンサーンとした．このアーキテクチャに基づく開発プロセスを定義した．紙幣搬送システムを事例として，設計したアーキテクチャの妥当性について考察した．信頼性の保証を目的とし，形式手法を用いてハードウェア間の同期が実現できることを確認した．また，開発の省力化を目的とし，アーキテクチャの実現としてのアプリケーションフレームワークの導入の可能性について考察した．

謝辞 本研究の一部は，科研費 (基盤研究 (C) 16K00110) および 2016 年度南山大学パッヘ奨励金 I-A-2 の助成による．

参考文献

[1] Gamma, E.: *Design patterns: elements of reusable object-oriented software*, Pearson Education India (1995).
[2] Hirschfeld, R., Costanza, P. and Nierstrasz, O.: Context-oriented programming, *Journal of Object Technology*, Vol. 7, No. 3 (2008).
[3] Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice Hall (1985).
[4] Noro, M. and Kumazaki, A.: On aspect-oriented software architecture: it implies a process as well as a product, *Software Engineering Conference, 2002. Ninth Asia-Pacific*, IEEE, pp. 276–285 (2002).
[5] Noro, M., Sawada, A., Hachisu, Y. and Banno, M.: E-AoSAS++ and its Software Development Environment, *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, IEEE, pp. 206–213 (2007).
[6] Shaw, M. and Garlan, D.: *Software architecture: perspectives on an emerging discipline*, Vol. 1, Prentice Hall Englewood Cliffs (1996).
[7] Tzilla, E., Robert E. Filman, Atef BaderShaw, M. and Garlan, D.: Aspect-oriented programming, Vol. 44, No. 10, CACM, pp. 29–32 (2001).
[8] 土居範久ほか：順路式，情報処理， Vol. 19, No. 8 (1978).