社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－SE－149（2）
2005／7／28

アスペクト指向ソフトウェアアーキテクチャスタイルの提案

熊崎 敦司[†]　野呂 昌満[†]

[†] 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

実現段階のアスペクトに関する研究は盛んに行われており，複数の計算モデルが提案されている．これらの計算モデルは相互に記述可能であるが，我々は一つの問題に対して一意の解記述を行えることが工学的に重要だと考えている．本研究では，計算モデル選択の規範を含む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提案する．既存の計算モデルを統一的に説明可能なアーキテクチャスタイルに基づき，選択の規範を示す．提案するスタイルにしたがったアーキテクチャ記述では，構成要素間の関係によって適切な計算モデルを選択できる．提案したスタイルで，携帯電話制御ソフトウェアのアーキテクチャを記述する例を通して，そのスタイルの有用性を確認する．

Aspect-Oriented Software Architecture Style

Atsushi Kumazaki[†], Masami Noro[‡]

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

There have been proposed several aspect-oriented computational models. One of the models can be interchangeably interpreted by another of the models. No concrete guidelines to select an appropriate computational model during a design process have been proposed. This paper proposed the aspect-oriented software architecture style(AOSAS) which gives the engineering guideline for selecting the computational model. A case study for the description of software architecture for mobile phone system shows the soundness of AOSAS.

1 はじめに

実現段階のアスペクト記述に関する研究は盛んに行われており，複数の計算モデル [4, 10, 13] が提案されている．アスペクト指向でソフトウェアを実現する場合，ソフトウェアの構造毎に適切な計算モデルが異なると考えられる．これらの計算モデルは相互に記述可能であり本質的な意味は同等であるが，どの計算モデルがどのような構造に適切であるかの指針は一般的に与えられているわけではない．我々は，一つの問題に対して一意の解が工学的（個人の能力に依存しない）に記述できることが重要だと考えている．すなわち，モデル選択の規範が必要であると考えている．

本研究では，計算モデル選択の規範を示す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提案する．既存の計算モデルを統一的に説明可能なアーキテクチャスタイルを基礎にし，構成要素間の関係により，適切な計算モデルの選択を可能にする．提案するアーキテクチャスタイルは，以下の特徴を持つ．

- アスペクト間記述を異なるアスペクトから抽出した構成要素群の協調場の記述と考える．

  既存の計算モデルにおけるアスペクト間記述は，アスペクトの合成方法を実現レベルで記述したものである．関連するアスペクトから抽出特定した構成要素同士の関係を記述することはアスペクト記述の文脈を与えることになる．この記述は実現レベルのものよりも抽象化されたものであり，アスペクトの仕様の一部と解することも可能である．

- アスペクトの階層を記述する．

  これまでのアスペクト指向開発経験 [5, 6] から，アスペクトの粒度は多様であることを認識している．異なる粒度のアスペクトを整理して記述するためにはアスペクトの階層化が必然である．すなわち，粒度の粗いアスペクトが粒度の細かいアスペクトから構成されることが確認できた．これは複数のコンサーンをグループ化したコンサーンが存在することを示し，直感と反するものではない．

本研究における問題解決の方針は以下の通りである．既存言語の計算モデルの抽象化を行い，統一的なアーキテクチャスタイルを提案する．このさい，アーキテクチャ段階のアスペクト，およびアスペクト間記述について考察する．さらに，構成要素間の関係とそれを実現するための適切な計算モデルとの関係を明らかにし，選択の規範について考察する．提案したアーキテクチャスタイルを用いて携帯電話制御ソフトウェアのアーキテクチャを記述する例を通して，以下を確認する．

- そのアーキテクチャスタイルが計算モデル選択の規範となること．
- アスペクトの文脈記述がアーキテクチャの理解に有効であること．
- アスペクトの階層化が構造整理に有効であること

2 関連研究

AOSAS（アスペクト指向ソフトウェアアーキテクチャスタイル）に関する研究も行われているが，多くは特定の計算モデルに基づくものである．これらとしては，オペレーションフックモデルを記述するもの [1, 12]，MDSoC モデルを記述するもの [2, 3]，独自のモデルを記述するもの [11]，等がある．これらは実現段階のソフトウェアの構造を記述するものであり，アーキテクチャの記述モデルではない．本研究では，アーキテクチャ段階でのアスペクトの記述について考察している．

アスペクト指向計算モデルの抽象化という観点では，増原らの研究 [7] がある．これは実現段階での計算モデルの抽象化を行い，各プログラミング言語が持つアスペクト指向の実現機構を整理したものと考えられる．言語段階での変換や容易な言語処理系の作成を可能としている．本研究では，アーキテクチャ段階での計算モデルの抽象化を行っている．これにより，異なるモデル記述を統一的に捉えることが可能である．

以上に加えて，本研究で提案するアーキテクチャスタイルは実現時の計算モデル選択の規範を示す．構成要素の関係に基づき，適切な計算モデルの選択を可能にする．さらに，アスペクト間記述としてアスペクトの文脈を記述している．これにより，各アスペクトおよびアスペクト間記述の理解度は向上する．

3 アスペクト指向ソフトウェアアーキテクチャスタイルの提案

計算モデル選択の規範をソフトウェアアーキテクチャで示すためには，まず，既存のアスペクト指向計算モデルを統一的に説明可能な AOSAS が必要となる．以下に，AOSAS 構築までの概略を示す．

- 既存のアスペクト指向計算モデルを抽象化した計算モデルを提案する．

既存の計算モデルを包括的に統合するのではなく，抽象化した計算モデルを提案する．これにより，わかりやすく単純な計算モデルを目指す．

- オブジェクト指向計算モデルとの親和性を考慮する．

  アスペクト指向技術をポストオブジェクト指向技術と捉え，提案するスタイルはオブジェクト指向計算モデルを拡張するものとする．

- 計算モデル選択の規範を示す．

  ソフトウェアアーキテクチャはプロセスを示唆する [8]，との考えに基づき，計算モデル選択の規範を示すことができるアーキテクチャを記述する．

- アスペクトの文脈を記述する．

  アスペクト間記述では，アスペクトの構成要素同士の関係とその順序やアスペクトの使用法を示すアスペクトの文脈を記述する．これにより，アスペクト，およびアスペクト間記述の理解が容易になる．

- アスペクトの階層を記述する．

  階層化は複雑なシステムを簡潔に表現するための一般的なアプローチである．粒度の細かいアスペクトを粒度の粗いアスペクトで包含することで，アスペクトの整理が可能である．

3.1 計算モデルの抽象化

オペレーションフック [4]，MDSoC[10]，ロールモデル [13] をアスペクト指向技術の代表と考え，これらを統一的に説明可能なモデルを考える．協調場をアスペクト間の関係記述と考えれば，ロールモデルも広義のアスペクト指向技術の一つと捉えることができる．横断的コンサーンを分離，記述したモジュールとそれらの関係記述という観点からアスペクト指向計算モデルを抽象化する．

オペレーションフックのモデルでは，横断的コンサーンにしたがって分割されたアスペクトの関係をオペレーションフックによって記述している．オブジェクト指向計算モデルを基に考えれば，アスペクトはオブジェクトの集合であり，オペレーションフックは，異なるアスペクトに属するオブジェクト間のメッセージ通信と解することができる．すなわち，オペレーションフックのモデルは，メッセージ通信の送信側としての合流点と合流点に付加するメッセージ通信をアスペクトと独立に記述するモデルである．

MDSoC のモデルでは，分割されたモジュールはハイパースライスであり，それらの関係はハイパーモジュールに合成のルールとして記述される．ハイパースライスはクラスの集合であり，ハイパーモジュールはハイパースライスの集合である．MDSoC のモデルは，クラス群として分割したハイパースライスをアスペクトとし，その合成の方法をアスペクトとは独立に記述するモデルである．

ロールモデルは，オブジェクト間の関係を協調場として独立に記述するものである．このモデルは，コンサーンにしたがって分割されたオブジェクト群の関係を協調場として独立に記述するモデルといえる．オブジェクト群をアスペクトと捉えれば，広義のアスペクト指向と考えられる．

以上のモデルを考察した結果，以下のように抽象化を行った．

- ソフトウェアは複数のモジュールから構成される．モジュールはオブジェクトの集合であり，それらの関係を記述したものである．
- モジュール間の関係は，あるモジュールのオブジェクトと別のあるモジュールのオブジェクト間の関係の集まりである．
- モジュール間の関係はモジュールとは独立に記述する．

構成要素

提案する AOSAS では以下の構成要素を用いてソフトウェアアーキテクチャを記述する．

- コンポーネント

  コンポーネントはオブジェクトに相当する．メッセージ通信のインタフェースを持ち，お互いにメッセージの送受信を行う．

- アスペクト

  アスペクトはあるコンサーンに基づいて分割したコンポーネントの集合であり，これらのコンポーネント間の関係も定義している．アスペクト内のコンポーネントは同じアスペクト内のコンポーネントとだけメッセージの送受信を行うことができる．アスペクトもコンポーネントである．

- アスペクト間記述

  アスペクトとアスペクトの合成関係を記述したものである．それぞれのアスペクトから関係のあるコンポーネントだけを抜き出し，それらの関

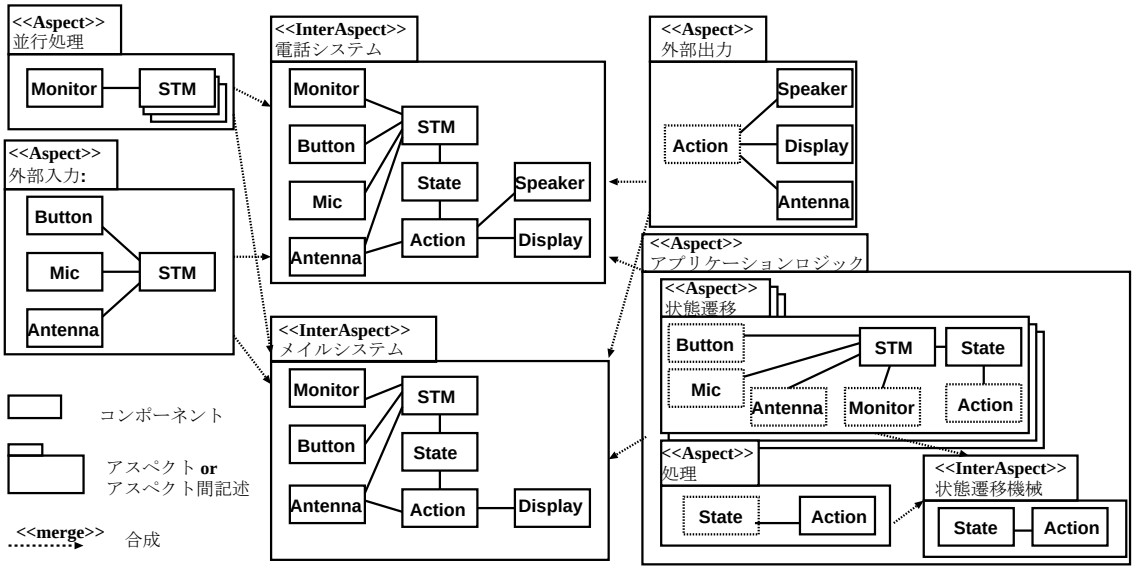

図 1: 携帯電話制御ソフトウェア

係（メッセージ通信）を記述する．アスペクト内のコンポーネントとアスペクト間記述内のコンポーネントの対応はコンポーネントの結合関係で記述する．

3.2 統一モデルによる記述

図 1 は携帯電話制御ソフトウェアの構成を記述したものである．この問題は単一の計算モデルでは，最適な構造を得られない例であり，適所に最適なモデルを適用している．携帯電話の機能として，通話機能，メイル機能を考えている．取り扱うハードウェアは，ボタン，画面，スピーカ，マイク，アンテナである．横断的コンサーンは，外部入力（イベント処理），外部出力，並行処理，実時間処理，例外処理である．

携帯電話制御ソフトウェアを状態遷移機械の集まりとして記述している．状態遷移機械（STM）は，ボタン（Button），マイク（Mic）等からの入力をイベントとして受け取り，対応する処理（Action）を行う．そのさい，携帯電話制御ソフトウェアは対話型のシステムなので，スピーカ（Speaker）やディスプレイ（Display）等を介して出力を行う．擬似並行実行モニタ（Monitor）で，複数の状態遷移機械を並行制御している．

アスペクト（<<Aspect>>）は，前述の横断的コンサーンを基にオブジェクト群を分割したものである．アスペクト間記述（<<InterAspect>>）はアスペクト内のコンポーネントを抽出し，それらの関係を記述したものである．通話機能を実現する場合と，メイル機能を実現する場合のコンポーネント，およびそれらの関係は異なるので，二つのアスペクト間記述がある．

図 1 は，UML[9] を利用して記述している．コンポーネントはクラス，アスペクト，アスペクト間記述はサブシステムを利用している．

オペレーションフックモデルの記述

例は携帯電話制御ソフトウェアにおいて，イベントの入力のさいに，並行処理のスケジューリングを行なうというものである（図 2 参照）．携帯電話制御ソフトウェアでは，複数の状態遷移機械が並行に動作しているものとしている．

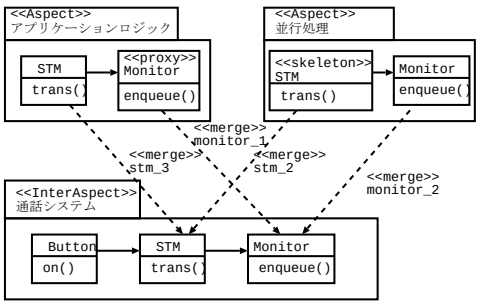

図 2: オペレーションフックの記述例

オペレーションフックを記述するさい，プロキシ，スケルトンの役割を持つコンポーネント（<<proxy>>，<<skeleton>>）を用いる．これらのコンポーネントは実際の処理は行わず，かならず他のコンポーネントと合成される．すなわち，これらのコンポーネントへのメッセージは合成した別のコンポーネントへのメッセージと捉えることができる．以上により，提案するモデルでは，コ

ンポーネントと，これらのコンポーネントとの関連を，オペレーションフックとみなす．これらのコンポーネントは本来，オペレーションフックのモデルでは存在しないが，モデルを抽象化し，プロキシ，スケルトンコンポーネントを導入することで，オペレーションフックのモジュール分割を，MDSoC の特殊形と捉えることができる．

この場合，アスペクト記述内の動的な挙動（メッセージの送信順序）が重要である．とくに，STM.trans と Monitor.enqueue の順序関係が記述されていなければ，アスペクトの合成を安全に行うことはできない．例えば，状態遷移機械はイベントを受け取ったさい，イベントを受理する前に並行処理のスケジューリングを行わなければならない．アスペクト間記述内の動的な挙動がわからなければ，意図した合成が行われるかの判断はできない．さらに，オペレーションフックの順序も重要になる．これについては後節で述べる．

MDSoC モデルの記述

携帯電話のアンテナが，外部入力コンサーンおよび，外部出力コンサーンによって分割された例である（図 3 参照）．外部入力アスペクトでは，アンテナの外部入力に関するメソッド (read メソッド) だけを記述し，外部出力アスペクトでは，アンテナの外部出力に関するメソッド (write メソッド) だけを記述する．アスペクト間記述（通話システム）では，通話システムに関連するコンポーネントとそれらの関係を定義している．そのさい，コンサーンによって分割されたオブジェクトの合成を行なっている．アスペクト間記述では，分割されたアンテナを合成している．

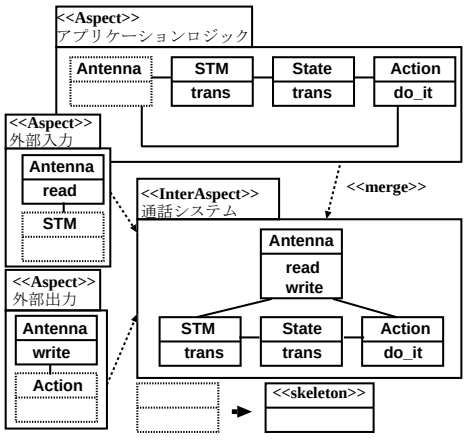

図 3: MDSoC の記述例

この場合，どのコンポーネントとどのコンポーネントが合成されて新しいコンポーネントができるかが重要である．コンポーネントの合成だけが記述されている場合には，合成されたコンポーネントが期待したコンポーネントであるかどうかの検査を行うことができない．合成するコンポーネントと合成後のコンポーネントを過不足なく記述することが必要である．

ロールモデルの記述

携帯電話のボタンが，それが属する協調場によって振舞いが変わるという例である（図 4 参照）．ボタンは，通話システムとしてのボタンおよび，メイルシステムとしてのボタンに動的に役割が変わる．役割の変化は協調場のコンポーネントとの結合関係の変化である．この場合，結合の順序関係を記述する必要がある．

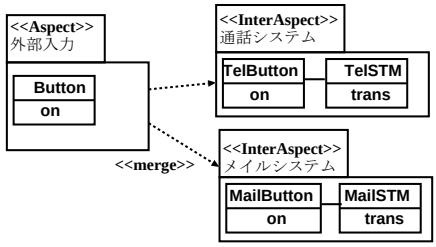

図 4: ロールモデルの記述例

アスペクトの文脈の記述

以上の議論を整理すると，静的な記述（MDSoC モデルの制約は静的な記述で可能である）に加えて，アスペクトの文脈として以下の動的な挙動を記述する必要がある．

- コンポーネントの結合関係
- コンポーネント間のメッセージ通信
- メッセージ通信と結合の順序

3.3 計算モデル選択の規範

計算モデルの選択を支援する方法として，“計算モデル選択時のプロセスを定義する” ことが考えられる．我々は，新しくプロセスを定義しなくても，アーキテクチャ記述がプロセスを示していると考えている [8]．提案したアーキテクチャスタイルでは，コンポーネントの関係が適切な計算モデル選択の規範になると考えている．

前節で示した各モデル記述におけるコンポーネントの関係について考察し，図 5 に示す規範を得た．結合元コンポーネントと結合先コンポーネントの関係が適切な計算モデルを示す．図 5 を基に統一モデルによる記述を観察すれば，適切な計算モデルを選択可能である．

3.4 アスペクトの階層化

ロールモデルの一種である Epsilon モデル [13] では，協調場内のロールと協調場の動的な結合を

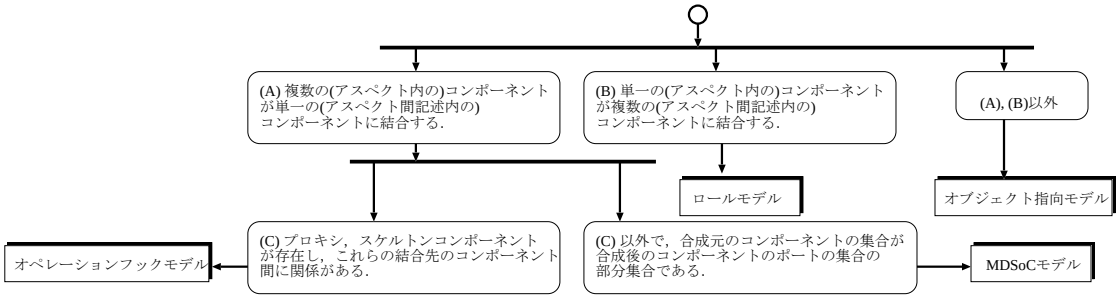

図 5: 計算モデル選択の規範

用いて，階層的に協調場を形成する．これにより，高い記述能力を得ている．しかし，反面，動的な構造の構築は挙動の把握に問題が生じることから，このモデルの実現言語 (Epsilon/J) においては，この機構をとり入れていない．

我々は以下の考えに基づき，アスペクトを階層的に記述する．

- 粒度の異なるアスペクト群を整理して記述するためにはアスペクトの階層化が必要である．
- 静的に階層構造を記述することにより，理解支援が可能になる．

携帯電話制御ソフトウェアのアプリケーションロジックはアスペクトを階層的に定義した例である（図 6 参照）．アプリケーションロジックアスペクトには，状態遷移に関するコンサーンと，そのさいに起こる挙動に関するコンサーンが存在する．これらをアスペクトとして分割して，状態遷移アスペクト，処理アスペクトとしている．また，それらの関係を定義したアスペクト間記述によりアスペクトを構成している．このように,(階層的)アスペクトは，複数のアスペクトと１つのアスペクト間記述によって構成される．

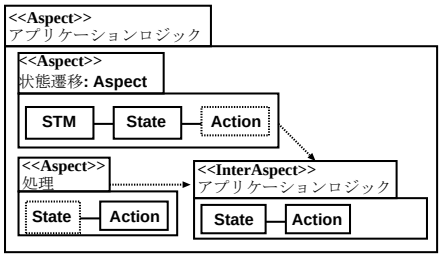

図 6: アスペクトの階層

4 考察

提案した記述モデルの有用性を確認するために以下の 3 つの観点から考察を行う．

- 提案した AOSAS が計算モデル選択の規範となるか．
- アーキテクチャの理解にアスペクトの文脈記述が有効であるか．
- アスペクトの階層化が構造整理に有効であるか．

4.1 計算モデル選択の規範に関する考察

オペレーションフックモデル，MDSoC モデル，ロールモデルを対象として，アーキテクチャ記述が計算モデル選択の規範となることを示す．

提案したモデルによるオペレーションフックモデルの記述例は図 2 に示している．この場合の特徴は以下の通りである．

- 合成されるアスペクトにそれぞれプロキシコンポーネント（<<proxy>>），スケルトンコンポーネント（<<skeleton>>）が存在する．
- これらのコンポーネントが合成したそれぞれのコンポーネント間に関係が存在する．

提案したモデルによる MDSoC モデルの記述例は図 3 に示している．この場合の特徴は以下の通りである．

- 複数の（アスペクト内の）コンポーネントが単一の（アスペクト間記述内の）コンポーネントと結合する．
- 合成元コンポーネントのポートの集合はそれぞれが合成後のコンポーネントのポートの集合の部分集合である．

提案したモデルによるロールモデルの記述例は図 4 に示している．この場合の特徴は，単一の（アスペクト内の）コンポーネントが複数の（アスペクト間記述内の）コンポーネントと結合することである．

図 5 に示した規範はこれらをまとめたものである．これを基に図 1 を観察すると，以下のことがわかる

- 以下の間の関係はオペレーションフックである．
  - –Monitor と STM
  - –Button, Mic, Anntena と STM
  - –Speaker，Display，Antenna と Action
  - –STM と Action
- Antenna は MDSoC モデルにより分割されている．

- 以下のコンポーネントは文脈によって振る舞いをかえる．
  –Button
  –Antenna
  –Display

このことは，ソフトウェアアーキテクチャが適切な計算モデルを示していると解することができる．

4.2 アスペクトの文脈記述に関する考察

図 7 は従来のオペレーションフックのモデル（AspectJ のモデル等）で携帯電話の通話システムを記述したものである．着信のさい，音と光で着信を知らせるもので，Action は Speaker と Dsipaly の両者にメッセージを送信する．この場合，以下の問題がある．

- 何のためにオペレーションフックを使用しているかがわかりづらい．
- オペレーションフック間の関係がわかりづらい．

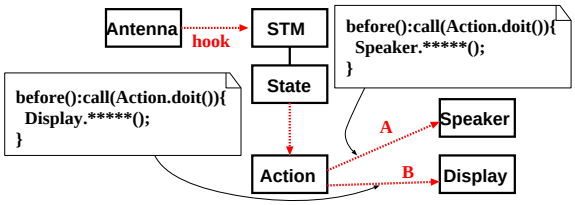

図 7: 従来のオペレーションフックモデル

図 8 は提案するモデルで前述の例を記述した場合のアスペクト間記述である．これにより，以下の利点を得ることができる．

- それぞれのアスペクトがどのような役割を担っているのかがわかる．
- オペレーションフック同士の関連がわかる．
- この文脈で必要なコンポーネント（または，アスペクト）がわかる．

その結果，前述の問題は解消される．

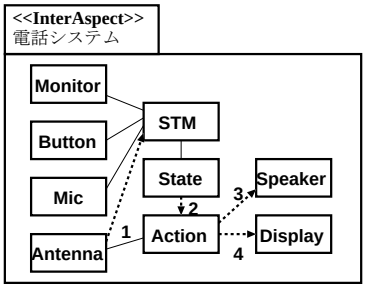

図 8: 文脈の記述（1）

携帯電話のアンテナは（電波の）入力と出力の両者の役割を持つ．また，最近では，光る等の機能もみられる．MDSoC の考えに基づけば，これらは独立に記述され，必要なものを合成して，適切なアンテナを実現することになる．しかし，それぞれのアンテナがどのようなコンサーンにしたがって分割したものかは記述されていないので，どのアンテナを合成すればよいかの判断が難しい．

提案するモデルでは，合成後のコンポーネントと合成関係をあわせて記述することで，合成を制限している（図 9 参照）．これにより，間違ったコンポーネントの合成や，合成すべきコンポーネントの不足等を知ることが可能である．

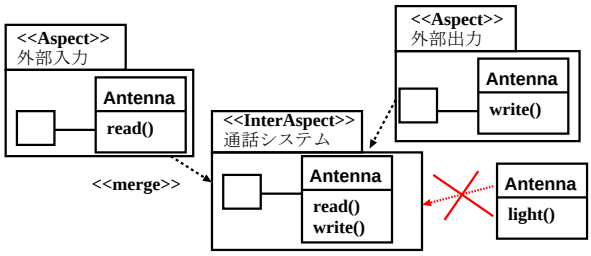

図 9: 文脈の記述 (2)

4.3 アスペクトの階層化に関する考察

既存の計算モデル [4, 10] ではアスペクトの階層化は考えられておらず，すべてのアスペクトを同一の粒度として記述していた．しかし，我々はこれまでのアスペクト指向開発経験 [5, 6] より異なる粒度のアスペクトが存在することを認識している．ソフトウェアの構造整理という観点から考えれば，これら粒度の異なるアスペクトは階層的に記述するのが適切だと考える．

図 6 は図 1 の一部であり，アプリケーションロジックを階層的に記述したものである．さらに，このアスペクトは他のアスペクト（並行処理，外部入力，外部出力）とも関連している．階層化を行わずに，同一の粒度のアスペクトとして記述すると図 10 のようになる．ただし，これはオペレーションフックのモデルを用いて記述している．

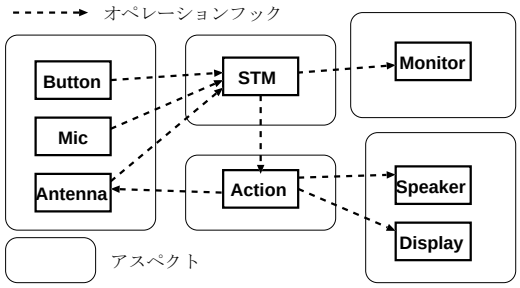

図 10: オペレーションフックモデルによる記述（階層化なし）

この場合の問題点は以下の通りである．

- STM，Action 間の分割の基となったコンサーンとそれ以外の分割の基になったコンサーンは直

交しており，STM，Action 間の分割と他の分割は意味が異なる．

- 現実モデルと隔たり（ギャップ）がある．

すなわち，現実には STM と Action により構成される状態遷移機械が他のアスペクトと関連している．しかし，このモデルではそれが記述できていない．以上の議論の結果，提案したスタイルのようにアスペクトを階層化したほうが自然な記述が可能であり，構造の理解が容易になるとの結論を得る．

5 おわりに

本研究では，実現時の計算モデル選択の規範を示す AOSAS を提案した．考察を通して，以下を確認した．

- 計算モデル選択の規範を示すソフトウェアアーキテクチャの記述が可能であること．
- アスペクトの文脈記述により，理解度が向上すること．
- アスペクトの階層化により構造の整理が可能なこと．

謝辞

この研究は,「2005 年度南山大学パッヘ研究奨励金 I-A」の援助を受けて行った．

参考文献

[1] W. M. Ho, J.M. Jezequel, F. Pennaneac'h and N. Plouzeau, "A Toolkit for Weaving Aspect Oriented UML Designs," *Proceedings of the 1st international conference on Aspect-oriented software development*, 2002.

[2] M. M. Kande, and A. Strohmeier, "On The Role of Multi-Dimensional Separation of Concerns in Software Architecture," *OOPSLA '2000 Workshop on Advanced Separation of Concerns*, 2000.

[3] M. Katara, and F. S. Katz, "Architectural views of aspects," *Proceedings of the 2nd international conference on Aspect-oriented software development*, 2003.

[4] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm and W.G. Griswold,"An Overview of AspectJ," *Proceedings of the European Conference on Object-Oriented Programming(ECOOP)*, Springer-Verlag. Hungary, 2001.

[5] 熊崎 敦司, 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 論文集, 2003.

[6] 熊崎 敦司, 野呂 昌満, 張 漢明, "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," 日本ソフトウェア科学会 FOSE2003 論文集, 2003.

[7] H. Masuhara and G. Kiczales, "Modeling Crosscutting in Aspect-Oriented Mechanisms," *Proceedings of ECOOP2003*, 2003.

[8] M. Noro, "On Aspect-Oriented Software Architecture:it Implies a Process as well as a Product," *Proceedings of APSEC2002*, 2002.

[9] OMG, *Unified Modeling Language Specification version 1.5*

[10] H. Ossher and P. Tarr, "Multi-Dimensional Separation of Concerns and the Hyperspace approach," *Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*, 2001.

[11] M. Pinto, L. Fuentes, and M. Troya, "DAOP-ADL : An Architecture Description Language for Dynamic Component and Aspect-Based Development," *Proceedings of the second international conference on Generative programming and component engineering*, 2003.

[12] D. Stein, S. Hanenberg, and R. Unland, "A UML-based Aspect-Oriented Design Notation for AspectJ", in *Proceedings of the 1st International Conference on Aspect-Oriented Software Development* , 2002.

[13] T. Tamai, N. Ubayashi and R. Ichiyama, "An Adaptive Object Model with Dynamic Role Binding," *Proceedings of International Conference on Software Engineering*, 2005.