アスペクト指向ソフトウェアアーキテクチャの図式表現に関する研究

M2004MM015 石川 智子
指導教員　野呂 昌満

1　はじめに

アスペクト指向ソフトウェアアーキテクチャを記述する方法として, 図式表現を用いるものとアーキテクチャ記述言語 (ADL) を用いるものに大別できる．UML に代表される図式表現は視覚的にアーキテクチャを表現するので直感的な理解を助ける. しかし, 図式表現のセマンティクスは一般的に曖昧な場合が多く, それによりアーキテクチャの複数の解釈ができてしまうことがある．一方 ADL は形式的な記述を用いてアーキテクチャの厳密な定義ができるが，直感的な理解という点では図式表現に劣る．以上からわかるように両者は補完的な関係にある．

アスペクト指向ソフトウェアアーキテクチャの図式表現に関する研究，ADL 記述に関する研究が行われている．しかし，双方を矛盾なく統一的な形式でアーキテクチャを表現するための方法は提案されていないと考える．

本研究の目的は UML を用いたアスペクト指向ソフトウェアアーキテクチャの図式表現を提案し，提案する図式表現のセマンティクスを記述することである．UML はソフトウェアのモデリング言語として広く普及し，受け入れられているので提案する図式表現の理解が得られやすいと考える．また, OCL が UML に組み込まれているので，OCL を用いて提案する図式表現のセマンティクスを厳密に記述し，曖昧さを補完できると考える．以上の理由から本研究では図式表現とそのセマンティクス記述に UML を採用する．

本研究ではアスペクト指向ソフトウェアアーキテクチャの題材として本研究室で提案されている，E-AOSAS++を取り上げる．アスペクト指向のモジュール記述は AspectJ と HyperJ で採用されている二つのモジュール記述に大別されている．E-AOSAS++は両モジュール記述を扱っており，アスペクト指向ソフトウェアアーキテクチャとして必要十分であると考え，これを採用する．

2　E-AOSAS++

本研究でアスペクト指向ソフトウェアアーキテクチャの題材とする E-AOSAS++について述べる．

XCC モデル

E-AOSAS++は本研究で提案されている組込みソフトウェアのソフトウェアアーキテクチャスタイルである．E-ASOSAS++は XCC モデルにしたがって構築される．XCC モデルではジェネリックアーキテクチャスタイル，ジェネラルアーキテクチャスタイルを規定している．

ジェネリックアーキテクチャスタイル

ジェネリックアーキテクチャスタイルはシステムはモジュール化されたコンポーネントとそれらの関係を定義するコネクタータイプと規定している．

ジェネラルアーキテクチャスタイル

ジェネラルアーキテクチャスタイルはジェネリックアーキテクチャスタイルにコンポーネントタイプとコネクタータイプを与えることで，より実現レベルの高いスタイルを規定している．

E-AOSAS++

ジェネラルアーキテクチャスタイルにモジュール分割のためのコンサーンを与え E-AOSAS++を規定する．

E-AOSAS++は組込みソフトウェアは並行に動作する状態遷移機械の集合と規定している．各並行状態遷移機械はイベントを送りあい，協調動作している．並行状態遷移機械は並行処理，状態遷移，コアの三つのアスペクトで構成される．アスペクト間の関係はアスペクト間記述を用いて実現される．また，状態遷移に関するアクションもアスペクト間記述を用いる．

3　アスペクト指向ソフトウェアアーキテクチャの図式表現

本節では E-AOSAS++を題材にアスペクト指向ソフトウェアアーキテクチャの図式表現の提案を行う．ジェネラルアーキテクチャスタイル構成要素の図式表現を提案し，その図式表現をもとに E-AOSAS++の構成要素を提案する．

3.1　ジェネラルアーキテクチャスタイルの図式表現

XCC モデルにおいてジェネラルアーキテクチャスタイルに与えられるコンポーネントタイプは原始コンポーネントと複合コンポーネントがある．また，コネクタータイプとしてコネクター，Unicast コネクター，Multicast コネクターがある．

原始コンポーネント

一級コンサーンによって分割されたシステムのモジュール単位．二級コンサーンによって分割されたアスペクトの集合で構成される．原始コンポーネントは UML-Component を用いて表現する． 各原始コンポーネントの役割と付加するステレオタイプを以下に示す． 原始コンポーネントの図式表現を図 1 に示す．

- Configuration Component
  (CComponent)
  システムの構成を実現する原始コンポーネント
  ステレオタイプ << *CComponent* >> を付加
- ConfigurationPolicy Component
  (CPolicyComponent)

CComponent を管理する原始コンポーネント
ステレオタイプ << *CPolicyComponent* >> を付加

- Aggregation Policy Component (APolicyComponent)
  ACComponent 内の原始コンポーネントを管理する原始コンポーネント
  ステレオタイプ << *APolicyComponent* >> を付加

アスペクトと IAD

アスペクトは二級コンサーンを実現するために協調して動作するオブジェクトの集合で構成される．UML の Collaboration を用いて表現する．ステレオタイプ << *aspect* >> を付加して定義する．

IAD はアスペクト間の関係を定義する．IAD を UML の Association を用いて表現する．アスペクトを織り込む場所，織り込まれる場所，織り込むタイミングを UML の Note を用いて Association に付加させる．Note にはアスペクト間記述を示すステレオタイプ << *IAD* >> を付加する．アスペクトと IAD の図式表現を図 1 に示す．

複合コンポーネント

複合コンポーネントは複数の原始コンポーネントの集合から構成される複合コンポーネントである．複合コンポーネントは原始コンポーネントと同様に UMLComponent にを用いて図式表現する．各複合コンポーネントの役割と付加するステレオタイプを以下に示す．図式表現を図 1 に示す．

- Configuration Composite Component (CCComponent)
  システムのコンフィギュレーションコントロールを実現．ステレオタイプ << *CCComponent* >> を付加
- Aggregation Composite Component (ACComponent)
  複数の原始コンポーネントの連動動作で実現する複合コンポーネント．
  ステレオタイプ << *ACComponent* >> を付加

コネクター

コネクターはコンポーネント間の関係を定義する．コネクターは IAD を用いて実現される．よってコネクターの図式表現は IAD と同様とする．

MulticastConnector

MulticastConnector は CCComponent に通知されたイベントを内部構成要素に通知し，IAD を用いて実現される．UML の DeligateConnector を用いて表現する．ステレオタイプ << *multicast* >> を付加して定義する．MultiConnector の図式表現を図 1 に示す．

UnicastConnector

UnicastConnector は ACComponent の内部構成要素の関係を定義し，アスペクト間記述を用いて実現される．UML の Connector を用いて表現する．ステレオタイプ << *unicast* >> を付加して定義する．UnicastConnector の図式表現を図 1 に示す．

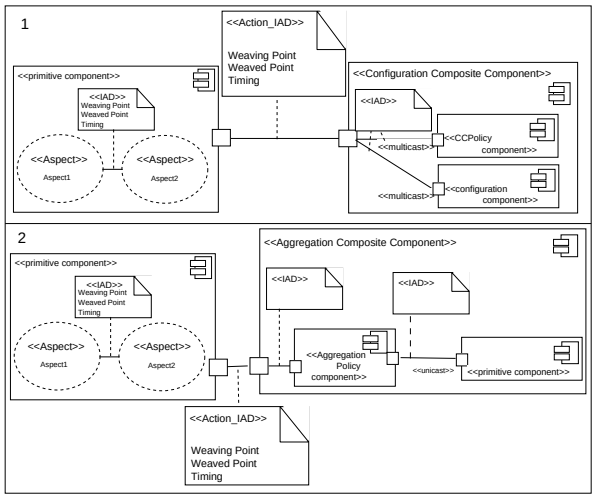

図 1: ジェネラルアーキテクチャスタイルの図式表現

3.1.1 ジェネラルアーキテクチャスタイル構成要素の抽象構文

前小節で図式表現を提案したジェネラルアーキテクチャスタイルの抽象構文を図 2 に示す．抽象構文を示すことにより，UML 構成要素と新しく提案したジェネラルアーキテクチャスタイル構成要素の関係を示す．

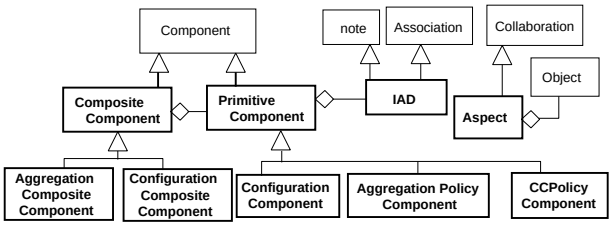

図 2: ジェネラルアーキテクチャスタイルの抽象構文

3.2 E-AOSAS++の構成要素図式表現

E-AOSAS++構成要素の図式表現は前小節で述べたジェネラルアーキテクチャスタイルの構成要素にステレオタイプを変更することで表現する．E-AOSAS++構成要素とジェネラルアーキテクチャスタイル構成要素と付加するステレオタイプの対応表を表 1 に示す．

動的挙動の図式表現

E-AOSAS++の動的挙動には UML のシーケンス図を用いる．アスペクト間記述と複合コンポーネント内コネクターにおける動的挙動の図式表現を図 3 示す．

アスペクト間記述は UML の Lifeline にステレオタイプ << *IAD* >> を付加することで定義する．また，UML の Message にステレオタイプ << *deligate* >> を付加し，実際の処理はアスペクト間記述が実現することを表現する．

複合コンポーネントにおけるコネクターは UML の複合フラグメント (並列) を用いる．並列複合フラグメントを

用いることで複数の処理が並列に実行されることを表現でき，ステレオタイプを付加して MulticastConnector，UnicastConnector を表現する．

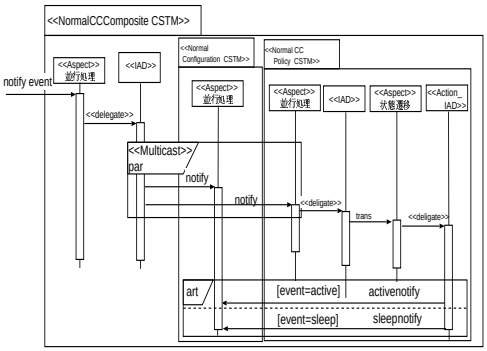

図 3: 動的挙動の図式表現

E-AOSAS++構成要素の抽象構文

抽象構文を示すことにより，UML 構成要素と新しく提案したジェネラルアーキテクチャスタイル構成要素の関係を示す．E-AOASA++構成要素の抽象構文を図 4 に示す．

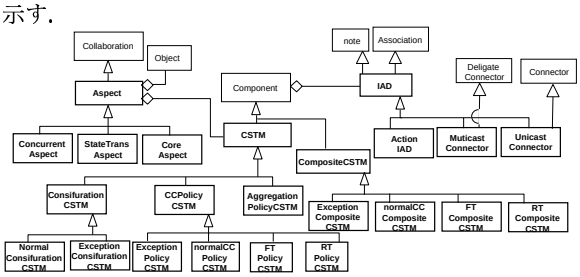

図 4: E-AOSAS の抽象構文

表 1 : 対応表

E-AOSAS++	ジェネラルアーキテクチャスタイル	付加するステレオタイプ
並行状態遷移機械	Primitive Component	
• Primitive CSTM	Primitive Component	<<PrimitiveCSTM>>
• Configuration CSTM	ConfigurationComponent	
– Normal Configuration CSTM	ConfigurationComponent	<<NormalConfigurationCSTM>>
– ExceptionConfiguration CSTM	ConfigurationComponent	<<ExceptionConfigurationCSTM>>
• CCPolicy CSTM	CCPolicy Component	
– RealTIme CCPolicy CSTM	CCPolicy Component	<<RTCCPolicyCSTM>>
– FaultTorerance CCPolicy CSTM	CCPolicy Component	<<FTCCPolicyCSTM>>
– ExceptionCCPolicy CSTM	CCPolicy Component	<<ExceptionCCPolicyCSTM>>
– AggregationPolicy CSTM	CCPolicy Component	<<APolicyCSTM>>
複合並行状態遷移機械	Composite Component	
• Configuration Composite CSTM	ConfigurationComposite Component	
– NormalCCComposite CSTM	ConfigurationComposite Component	<<NormalCCCompositeCSTM>>
– Exception Composite CSTM	ConfigurationComposite Component	<<Exception Composite CSTM>>
– Fault Torerance Composite CSTM	ConfigurationComposite Component	<<FTCompositeCSTM>>
– RealTime Composite CSTM	ConfigurationComposite Component	<<RTCompositeCSTM>>
• Aggregation Composite CSTM	AggregationComposite Component	<<AggregationCompositeCSTM>>
Aspect	Aspect	
• Concurrent Aspect	Aspect	<<ConcurrentAspect>>
• StateTrans Aspect	Aspect	<<ST Aspect>>
• Core Aspect	Aspect	<<Core Aspect>>
アスペクト間記述	Connector	<<IAD>>
• Action	Connector	<<Action_IAD>>
• Muticast Connector	Muticast Connector	<<multicast>>
• Unicast Connector	Unicast Connector	<<unicast>>

4 アスペクト指向ソフトウェアアーキテクチャのセマンティクス記述

本節では UML を用いたアスペクト指向ソフトウェアのセマンティクス記述に関して述べる．UML の OCL とシーケンス図を用いてセマンティクスを記述する．以下にそれぞれの役割を示す．

- OCL
  静的セマンティクスを記述
  → 内部構成要素を持つ構成要素の構造を定義
- シーケンス図
  動的セマンティクスを記述
  → 構成要素間の関連を定義

通常 ConfigurationCSTM を例に静的，動的セマンティクス記述を示す．

静的セマンティクスの記述例

```
Context NormalConfiguration CSTM inv;
  self.oclsTypeOf(CTM) implies
    //通常ConfigurationCSTMは並行状態遷移機械であり，
  self.ownedElement =ConcurrentAspect,StateTransAspect,CoreAspect
    //構成要素は並行処理,状態遷移,コアアスペクトである.
  self.owner = ConfigurationCompositeCSTM
    //ConfigurationCompositeCSTMの構成要素である.cc
```

動的セマンティクスの記述例

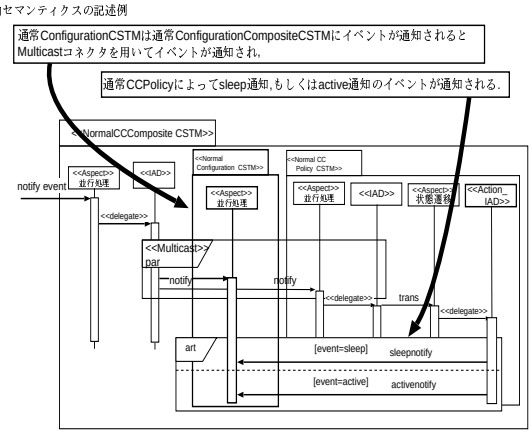

図 5: セマンティクスの記述例

5 考察

本節では提案した図式表現とそのセマンティクス記述に関する考察を行う．

5.1 提案した図式表現の妥当性に関する考察

コンポーネントとアスペクトの図式表現と動的挙動の図式表現の妥当性を議論する．

原始コンポーネントと複合コンポーネントの図式表現の妥当性に関する考察

コンポーネントの図式表現として Package と Component などの UML 構成要素が考えられる．各 UML 構成要素を用いたコンポーネントの図式表現を図 6 に示す．両 UML 構成要素とも分類子をグループ化するものだが，Package は外部内部の概念がなく，複合コンポーネントの UnicastConnector を表現することができない．

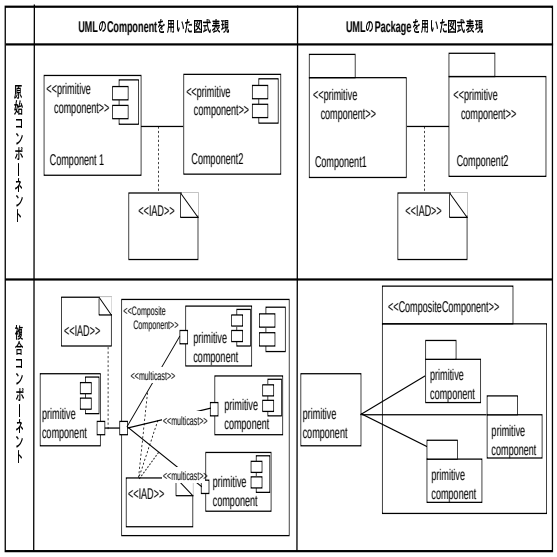

図 6: コンポーネントの図式表現

よって，コンポーネントの図式表現には UMLComponent を採用する．

アスペクトの図式表現の妥当性に関する考察

アスペクトの図式表現として Component，Package，Class，Collaboration の UML 構成要素が考えられる．UMLComponent を用いてアスペクトを表現する場合，コンポーネントと同様の図式表現となり，アスペクトとコンポーネントの違いがわかりにくくなってしまう．Component の内部構成要素は分類子であり，Package は分類子ではないので Package を用いたアスペクトの図式表現はできない．Class を用いてアスペクトを表現するとアスペクトとアスペクトの構成要素であるオブジェクトとの違いがわかりにくくなってしまう．Collaboration はある役割を協調して果たす分類子の集合を表現するのでアスペクトを表現可能であり，上記したような不具合が起こらないと考え，これを採用する．アスペクトの図式表現を図 7 に示す．

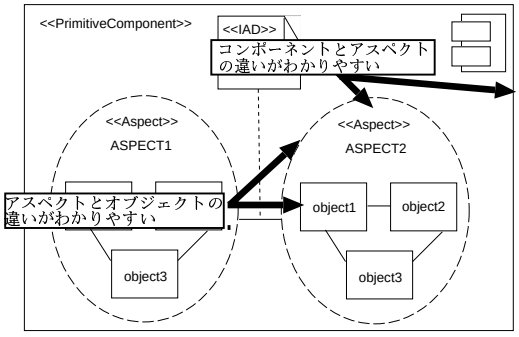

図 7: アスペクトの図式表現

動的挙動の図式表現の妥当性に関する考察

ソフトウェアの動的挙動を表現する図式表現はシーケンス図とコミュニケーション図の二つある．

どちらの記述法を用いてもソフトウェアの動的挙動を表現することはできる．しかし，コミュニケーション図はメッセージの順序を番号のみで表現するのでわかりにくい．また，時系列よりもオブジェクト間のやりとりを重視して表現している．ソフトウェアの静的な構造はクラス図を用いて表現することができるので，ソフトウェアの動的な挙動はシーケンス図を用いて表現する．

5.2 UML を用いたセマンティクス記述に関する考察

本研究では提案した図式表現のセマンティクスを UML を用いて記述した．静的セマンティクスを OCL を用いて，動的セマンティクスはシーケンス図を用いて記述した．ここでいう静的セマンティクスは構成要素の内部構成要素を示し，動的セマンティクスは構成要素間の関連を示す．

OCL を用いて構成要素の内部構成要素を不変条件として記述することができた．シーケンス図を用いて構成要素間の関連を記述することができた．以上から UML を用いたセマンティクス記述が可能だと考える．

5.3 アスペクト指向ソフトウェアアーキテクチャに関する考察

アスペクト指向モジュール記述モデルは AspectJ[1] と HyperJ[2] の両モジュール記述モデルに大別される．本研究では AspectJ で採用されるアスペクト間記述の機構を UML の Association と Note を用いて図式表現した．また，HyperJ における HyperSlice と HyperModule の関係は複合コンポーネントに反映されていると考える．以上の理由から，両モジュール記述モデルの図式表現として必要十分と考える．

6 おわりに

本研究ではアスペクト指向ソフトウェアアーキテクチャの図式表現とそのセマンティクスを UML を用いて記述した．UML を用いてセマンティクスを記述することで UML のツールがソフトウェアのセマンティクスを理解することができ，アスペクト指向ソフトウェア開発のツールサポートの可能性を与えることができたと考える．今後の課題としては提案した図式表現の実例への適用，ツールを用いた実例検証が考えられる．

参考文献

[1] “AspectJ Project,” *http://aspectj.org/*, 2006.

[2] H. Ossher, and P. Tarr “Using Multidimentional Separation of Concerns to (Re)Shape Evolving Software,” *CACM*, 2001 , vol. 44, pp. 43-50.

[3] OMG“UML2.0 superstructure specification,” *http://www.omg.org/*,2005.

[4] J. Warmer, A. Kleppe, “Object Constraint-Language,” *Addison Wesley Longman,Inc* , 1999.