アスペクト指向ソフトウェアアーキテクチャに基づいたソフトウェア開発の支援に関する研究

2000MT044 小出 資久 2000MT076 老子 隆之
指導教員　野呂 昌満

1　はじめに

現在我々の研究室では，アスペクト指向ソフトウェアアーキテクチャスタイル (Aspect-Oriented Software Architecture Style, 以後 AOSAS)[4] を提案している．AOSAS に基づいたアスペクト指向ソフトウェアアーキテクチャ (Aspect-Oriented Software Architecture, 以後 AOSA) の実現において，メッセージフックを拡張したモデレータフレームワーク [1] を用いて記述する．モデレータフレームワークを用いた開発では，定型コードが増え開発の労力が増える．

図式表現からコードを生成する開発支援は，古くから研究課題として考えられている．近年では，UML を利用し CASE ツールを用いた開発支援が行われている．アスペクト指向は新しい開発技術である．言語の設計に関する研究は行われているが，アーキテクチャの記述方法や開発支援に関する研究は盛んには行われていない．本研究では従来の図式表現からコードを生成する開発支援が開発の省力化になるという仮定にたちアスペクト指向開発への適用を試みる．

本研究の目的は，図式表現からコードを生成する開発支援をアスペクト指向開発に適用し，開発の省略化に有効であるかを考察することである．

コード生成系の入力には，現在我々の研究室で提案されている AOSA の図式表現の内部表現を利用する．AOSA の図式表現とは，AOSA の図式と，アーキテクチャに基づいた実現段階の図式を指す．出力は，モデレータフレームワークのコードと AspectJ[2] のコードとする．

図式表現には誤りが記述されている可能性があるので，図式表現の内部表現の検査を行う．検査としては，字句の検査や型検査に加えて，メッセージフックの検査を行う．AspectJ の処理系ではジョインポイントに関して厳密に検査を行うことができない．また，モデレータフレームワークを使用した場合，Java 言語の処理系を用いているのでメッセージフックについて検査を行うことができない．これらの検査を行うことで図式表現の誤りが早期発見でき，生成後の誤りが減少すると考える．

実際に電子辞書のアプリケーションを実現し，コードを生成することで以下のことが確認できた．

- クラスのインタフェースの記述部分とモデレータフレームワークのホットスポットが生成できた．
- 字句や型の誤りや，メッセージフックに関する誤りが減少した．

アスペクト指向開発において，本研究で試作したコード生成系を用いた開発支援が有効であることを確認した．

小出は内部表現の検査について，老子はコード生成系を主に担当した．

2　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

本研究で扱う AOSA の図式表現は, 以下の 2 つの図式によって構成されている [5].

- AOSA の図式
- アーキテクチャに基づいた実現段階の図式

AOSA の図式は，AOSAS の構成要素であるフィールド，ロール，オブジェクトを記述する．アーキテクチャに基づいた実現段階の図式では，実際に実現するオブジェクトを記述する．AOSA の図式表現は，UML エディタの拡張機能を利用している．UML エディタの機能を利用して図式表現の内部表現を生成する．内部表現を変換系の入力として ASAX が生成される．ASAX(Aspect-Oriented Software Architecture in XML) は，AOSA の図式表現を XML[3] で表現したものである．

3　コード生成系と ASAX の検査

図式表現からコードを生成する手順を**図 1** に示す．手順は以下の通りである．開発者は UML エディタを用いて AOSA の図式表現を記述する．UML エディタの機能を用いて XMI を生成し，XMI と XMI の DTD(Data Type Difinition) を変換系に与えて ASAX を生成する．コード生成系の入力には，ASAX と ASAX の DTD を用いる．ASAX の構造の定義には，現在標準である DTD を用いる．コード生成系の出力は，アスペクト指向言語として一般的である AspectJ と広く普及している Java 言語で記述したモデレータフレームワークのコードを生成する．本研究では，**図 1** の枠で囲まれた部分を扱う．

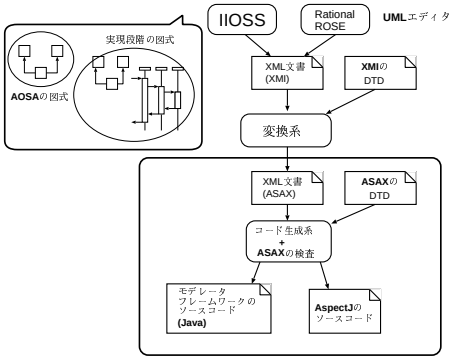

図 1　図式表現からコードを生成する手順

3.1　ASAX の検査

コード生成系の入力である ASAX に誤りが含まれていると，生成後のコードをコンパイルしたさいにエラーが生じる場合がある．本研究では，コード生成時に ASAX の検査を行うことで生成後のエラーを減らし労力削減を行う．

検査の定義

ASAX の検査には，字句の検査，意味の検査を行う．構文の検査は，DTD で ASAX の構造を定義することができ

る．字句の検査では，識別子の検査を行う．意味の検査では，メッセージフックに関してクラスやメソッドなどの宣言の有無の検査と型検査を行う．

- 字句の検査
  生成系の出力としてモデレータフレームワークとAspectJ のコードを生成するので，Java 言語において使用してはいけない識別子について検査を行う．変数名やメソッド名などについて，1 文字目は英字 (A〜Z，a〜z)，アンダースコア (_)，$ のいずれかであること．2 文字目以降は，数字と 1 文字目で使用できる文字であること．また，Java 言語で定められている予約語を識別子として利用することはできない．
- 意味の検査
  メッセージフックのジョインポイントやフック先のクラスや実行されるメソッドが正しく指定されていないと生成後のコードをコンパイルしたさいにエラーが出力される．AspectJ の処理系では，メッセージフックに関してジョンポイントやアドバイスについて検査を行っている．モデレータフレームワークを使用した場合は Java 言語の処理系を用いているので，AspectJ 相当の検査ができない．従って，ASAX の検査を行うことで生成後のコンパイルエラーが減少すると考える．メッセージフックの記述からジョインポイント先のクラスやメソッド，メソッドの返り値の型，引数の型が宣言されているかどうか，またアドバイス処理の記述からフック先のクラスや実行されるメソッドが宣言されているかどうかの検査を行う．

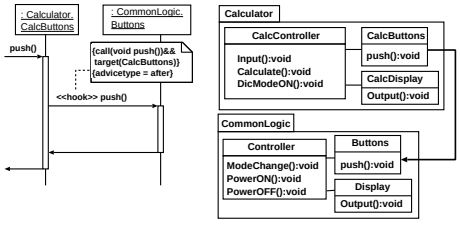

図 2　AOSA の図式表現とメッセージフックの例

	メッセージフックの記述	ジョインポイント先のクラス	実行されるアドバイスの記述	フック先のクラス
クラス名	**target** で指定しているクラス名	図式表現でのクラス名	指定しているクラス名	図式表現でのクラス名
メソッド名	**Call** で指定しているメソッド名	図式表現でのメソッド名	指定しているメソッド名	図式表現でのメソッド名
メソッドのリターンバリューの型	**Call** で指定しているメソッドのリターンバリューの型	図式表現でのメソッドのリターンバリューの型		
メソッドの引数の型	**Call** で指定しているメソッドの引数の型	図式表現での引数の型		

図 3　意味の検査の対応

意味の検査の対応について**図 2** と**図 3** を用いて説明する．**図 3** でのジョインポイント先のクラスは**図 2** の CalcButtons クラス，**図 3** でのフック先のクラスは**図 2** の Buttons クラスである．意味の検査の対応は**図 3** のメッセージフックの記述とジョインポイント先のクラスのそれぞれの部分について行う．また，実行されるアドバイスの記述とフック先のクラスについても同様に検査を行う．

型検査については，ASAX においてプログラミング言語の基本データ型の情報が全て定義されるので，この基本データ型の情報を記号表と考える．記号表を参照することで型検査を行う．

3.2　コード生成系

本研究で試作したコード生成系は，入力に ASAX を用いる．出力は，AspectJ のコードとモデレータフレームワークのコードとする．コード生成系では，抽象構文木を作成し木を操作するので，広く普及してる XML パーサである DOM パーサを用いる．

3.2.1　ASAX からモデレータフレームワークを生成するコード生成系

モデレータフレームワークを用いてメッセージフックを実現することで言語に依存しない構造を得ることができる．しかし，モデレータフレームワークを用いて実現を行うと多くのクラスを生成しなければならなくなり，開発者の労力となる．コード生成系を試作することで労力削減を行う．

ASAX と生成されるクラスの対応について

生成される全てのクラスを対応づけることができるが，同じような対応関係になるので，メッセージフックのためのモデレータフレームワークは WithAspect クラスについて対応関係を示す．**図 4** に生成されたモデレータフレームワークのクラス図を示す．

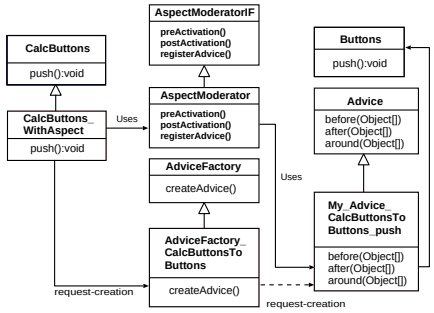

図 4　生成されたモデレータフレームワーク

CalcButtons_WithAspect クラスを生成する時の対応関係を**図 5** に示す．

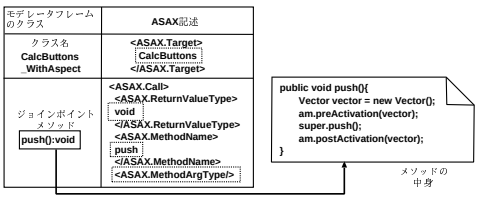

図 5　CalcButton_WithAspect クラスの生成について

ジョインポイントとなるメソッドは ASAX の ASAX.MethodName で指定されたメソッドである．また，メソッドの返り値の型は ASAX.ReturnValueType で指定された型，メソッドの引数の型は ASAX.MethodArgType で指定された型となる．ジョインポイントとなるメソッドの前後にメッセージを付加することでメッセージフック行う．

実現したコード生成系

実現したコード生成系は，ASAX の検査を行うクラスとコード生成を行うクラスがある．ASAX の検査を行う Check クラスでは，字句の検査と意味の検査を行う．コード生成を行うクラスは，Visitor パターンを用いることで，ノードをたどるクラスと処理を行うクラスを分離する．実現したコード生成系のクラス構成を**図 6** に示す．

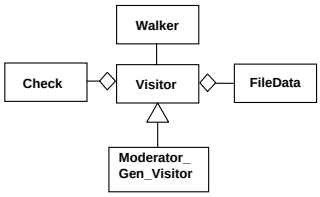

図 6　生成系のクラス図

3.2.2　ASAX から AspectJ を生成するコード生成系

AOSA に従ってコードを作成するとメッセージフックの記述が多くなり，開発者の労力となる．メッセージフックの記述を生成することで開発者の労力削減を行う．対応関係を考えることがコード生成系の設計に関係するので，ASAX と生成するアスペクト記述の対応関係を説明する．コード生成系はモデレータフレームワークを生成するコード生成系の Visitor クラスのサブクラスをカスタマイズすることで実現することができるので，説明とクラス図は省略する．

ASAX と生成されるアスペクト記述の対応について

入力となる ASAX は**図 2** を基に作成されたものである．ASAX の記述と生成するアスペクト記述の対応関係を**図 7** に示す．

	ASAX	生成されるアスペクト記述
アスペクト名	<ASAX.Target> <ASAX.MethodName> <ASAX.Body>	Aspect CalcButtons ToButtons_push
ポイントカット名	<ASAX.PointcutMethodName> <ASAX.PointcutArgument>	pointcut atCallpush() :
ジョインポイント	<ASAX.Target> <ASAX.Call> <ASAX.ReturnValueType> <ASAX.MethodName> <ASAX.ArgType>	target(CalcButtons) && call(void push());
アドバイス	<ASAX.Before> <ASAX.After> <ASAX.Around> <ASAX.UserDefinedPointcutName> <ASAX.UserDefinedPointcutArgument>	after() : atCallpush()
アドバイスの処理記述	<ASAX.Body>	Buttons.getInstance().push();

図 7　ASAX とアスペクト記述の対応

生成されるアスペクト記述は**図 7** のような ASAX の要素の情報を用いることで生成することができる．

4　考察

この節では，ASAX の検査の有効性とコード生成系の有効性について考察する．ASAX の検査，コード生成系の有効性を議論するための具体例として，AOSAS に基づいた電子辞書を試作した．試作した電子辞書の AOSA の図式を**図 8** に示し，実現段階の図式を**図 9** と**図 10** に示す．

4.1　ASAX の検査の有効性

ASAX の検査として，字句の検査・意味の検査やメッセージフックに関する検査について述べ，ASAX の有効性について議論する．

字句の検査では, 識別子の検査を行った．識別子の検査は，コード生成系に Java の API を用いて正規表現を含んだ処理を記述することで行うことができた．また，変数名やメソッド名に予約語の使用を検査する処理も含めることで識別子の検査を行うことができる．**図 9** の CalcButtons クラスのメソッド名に 3push のようなメソッド名を記述していた場合は，生成を中止する．

意味の検査は，メッセージフックのジョインポイントに指

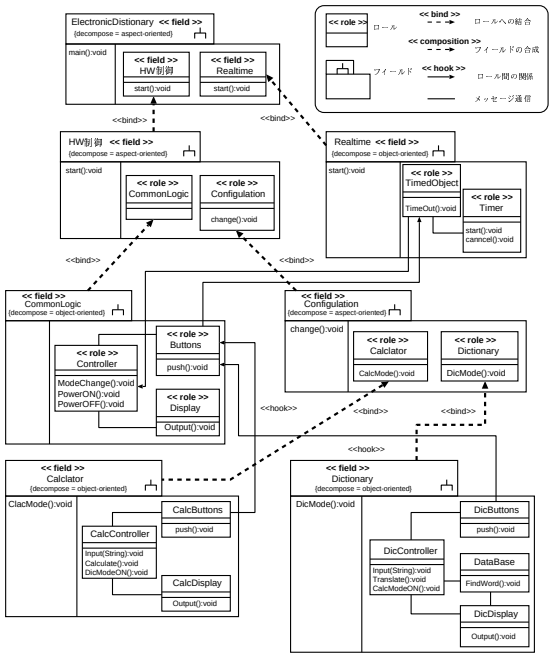

図 8　電子辞書の AOSA の図式

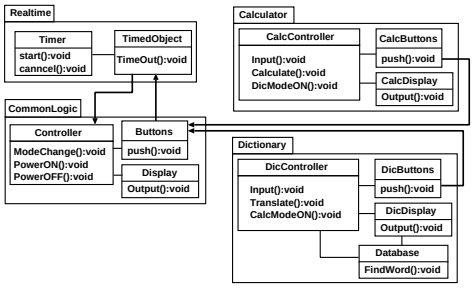

図 9　電子辞書の実現段階の図式：クラス図

定しているクラス，メソッド，メソッドの返り値の型，メソッドの引数の型やフック先のクラス，実行するメソッドについて宣言されているかどうかの検査と型検査を行った．メッセージフックについての検査の例として，**図 10** に示されている Calculator フィールドの Buttons クラスの push メソッドをジョインポイントとし，実行されるメソッドを CommonLogic フィールドの Buttons クラスの push メソッドとしたメッセージフックの場合を考える．ジョインポイントとして指定しているクラス (Calculator フィールドの Buttons クラス) は宣言されているが，メソッド (CommonLogic フィールドの Buttons クラスの push メソッド) が存在しない場合は，生成時にエラーを出す．型検査は，ASAX にプログラミング言語の基本データ型の情報が含まれているので，これを記号表と考え，型検査を行う要素の属性値と記号表を照らし合わせることで型検査を行うことができた．

DTD では要素の内容について制限を加えることができない．また，正規表現を使うことができない．現在，事実上標準となりつつある XML Schema では，要素の内容について制限を加えることができ，正規表現を使うことができる．従って，識別子の検査は XML Schema を用いて正規表現を定義することで行うことができる．予約語の使用の検査や意味の検査については，DTD を用いた場合と同様の検査を行うことで検査できる．

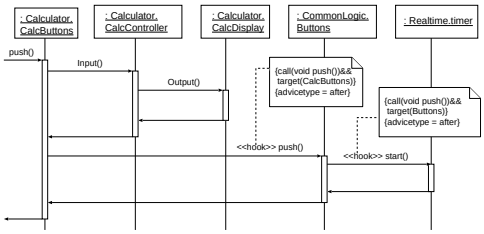

図 10　電卓モードで「3」を出力する場合の実現段階の図式：シーケンス図

AspectJ の処理系は，アスペクトを Java のコードに織り込む作業と織り込んだコードをコンパイルするという作業を行っている．ジョインポイントの指定にアスタリスク (ワイルドカード) を用いることができるので，アスペクトを織り込む作業をするときに厳密な検査を行うことはできない．指定したジョインポイントが実際に存在しない場合は，アスペクトを Java のコードに織り込まないので，コンパイル時にコンパイルエラーとして出力されることはない．開発者は実行時になって始めて意図した通りにプログラムが動かないことがわかり，そこからデバッグを開始しなければならない．本研究でコード生成系の入力とする ASAX は，ジョインポイントで指定しているクラスやメソッドについて，またアドバイスで実行するクラスのメソッドの情報が含まれている．またアスタリスクは考えない．従って，コード生成時にメッセージフックに対して検査を行うことが可能である．検査を行い，エラーを出力するので実行時にメッセージフックに関して開発者がデバッグを行う必要がなくなる．

モデレータフレームワークは，既存の処理系ではメッセージフックについてのコンパイルエラーが出力されずに，クラスが存在しないというエラーが出力されるだけである．本研究での検査を行うことで，生成時にメッセージフックに関するエラーが出力される．

本研究で定義した検査を行うことによって，生成時に言語に依存している部分の検査を行うことができた．検査を行うことによって，生成したコードに関して生成後や実行時の誤りが減少することを確認した．

4.2　コード生成系の有効性

AOSA を実現する方法として，モデレータフレームワークを用いる方法と AspectJ を用いる方法について考える．それぞれを AOSA に従いコード生成系を用いないで実現を行う場合と図式表現を用いて ASAX を作成し，コード生成系を用いて実現を行う場合とを比較することで，コード生成系の有効性について議論する．

モデレータフレームワークを用いた実現を考えた場合

モデレータフレームワークを用いてメッセージフックを行う場合を考える．コード生成系を用いない場合開発者は，**図 9** に示される全てのオブジェクトを記述し，モデレータフレームワークのホットスポットも記述しなければならない．本研究のコード生成系はモデレータフレームワークのホットスポットは全て生成できた．また，オブジェクトに関してはクラスのインタフェースを生成できるので，開発者はオブジェクトに必要な処理を書き加えるだけでよくなる．また，検査を行うのでコード生成後のコンパイルエラーが減少する．

AspectJ を用いた実現を考えた場合

AspectJ を用いてメッセージフックを行う場合を考える．コード生成系を用いない場合開発者は，**図 9** に示される全てのオブジェクトを記述し，AspectJ によるメッセージフックも記述しなければいけない．コード生成系を用いた場合，オブジェクトはクラスのインタフェースだけを生成できた．また，AspectJ によるメッセージフックは全て生成できた．

二つのコード生成系を試作した．二つのコード生成系ともにメッセージフックの記述，クラスのインタフェースが生成でき開発者はオブジェクトに処理を書き加えるだけでよくなる．また，コード生成時に ASAX の検査をすることで生成されたソースコードに関してはエラーがなくなることを確認した．従って，本研究のコード生成系はアスペクト指向開発において，十分に開発の支援に有効であると確認した．

5　おわりに

本研究では，図式表現からコードを生成する開発支援をアスペクト指向開発に適用し，開発の省力化に有効であるかを考察した．試作したコード生成系を用いることでクラスのインタフェース部分とモデレータフレームワークのホットスポットが生成できた．生成されるコードのプログラミング言語の字句と意味に適応するかどうかの検査 (識別子の検査，メッセージフックについての検査，型検査) を行うことで生成後や実行時の誤りが減少することを確認した．これによりコード生成系がアスペクト指向開発の支援に有効であることを確認した．

今後の課題として，ASAX から他のアスペクト指向言語のコードを生成することを考える．

謝辞

本研究を進めるにあたり，二年間御指導いただいた野呂昌満教授，有益なアドバイスをいただいた張漢明助教授，蜂巣吉成講師，親身になって相談にのってくださった大学院生の熊崎敦司さん，森貴彦さん，藤原泰昌さん，後藤修平さんに深く感謝致します．

参考文献

[1] C.Constantinides,A.Bader. T.Elrad and M.Fayad: Designing an Aspect-Oriented Framework in an Object-Oriented Environment, *ACM Computing Surveys,2000.*

[2] http://eclipse.org/aspectj, AspectJ Aspect-Oriented Programming.

[3] http://www.w3.org/TR/REC-xml,T.Bray,J.Paoli,C.M.Sperberg-McQueen and E.Maler: World Wide Web Consortium (W3C), eXtensible Markup Lan-guage(XML)1.0(Second Edition), 2000.

[4] 熊崎敦司: 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その実現に関する研究，南山大学経営学部情報管理学科博士論文,2004.

[5] 森貴彦: 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，南山大学経営学部情報管理学科修士論文,2004.