XML 文書処理系のアーキテクチャの提案
– デザインパターンを用いた DOM 木の処理 –

2005MT027 市川 俊太　　2005MT050 川上 智大
指導教員　蜂巣 吉成

1　はじめに

XML を利用した情報交換が注目され，XML データを処理するアプリケーションが開発されている．XML では DTD などのスキーマ言語によって XML ボキャブラリを定義することで XML 文書を記述することができる．XML 文書を解析できる DOM パーサが既に提案されており，XML データを処理するアプリケーションは DOM 木を利用することが多い．DOM 木の探索処理，ノードの照合処理などは DOM パーサの利用者である XML 文書処理系開発者が記述する．しかし，DOM 木の処理に関する具体的な記述方法について確立されていないので，アプリケーション開発の手間が大きく，既存のアプリケーションの修正や再利用も難しい．
本研究では DOM 木の処理の記述方法を明確にするために，デザインパターンを用いて XML 文書処理のアーキテクチャの提案を行う．提案したアーキテクチャに基づいて DOM 木を処理する開発手順を提示することで XML 文書処理系の開発を支援する．

2　DOM 木処理とその問題点

2.1　XML 文書と DOM 木の関係

DOM パーサで XML 文書を読み込むと，図 1 のような DOM 木と呼ばれる木構造のデータを作成する．木のノードは要素，テキスト，属性などである．DOM の API を利用することで木を走査できる．図 1 の XML 文書の例は雇用者名簿に関するデータを表している．

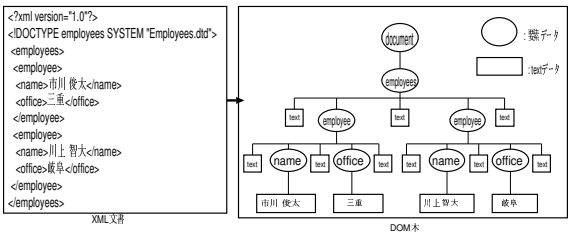

図 1　XML 文書と DOM 木

2.2　一般的な DOM 木処理

DOM に基づくプログラムでは，木構造のノードを順にアクセスする処理，特定の条件にマッチするノードの照合を行う処理，アプリケーション固有の処理などを記述する．また，ノードなどを集約したノードの集合に対する操作を行う処理が考えられる．DOM 木の一般的な処理は，次の 4 つの処理に分類できる．

- 木の走査
- ノードの照合
- ノードの集合に対する操作
- アプリケーション固有の処理

DOM 木処理の流れを図 2 に示す．木の走査でノード毎にノードの照合を行い，ノードの照合から対応したアプリケーション固有の処理を行う．

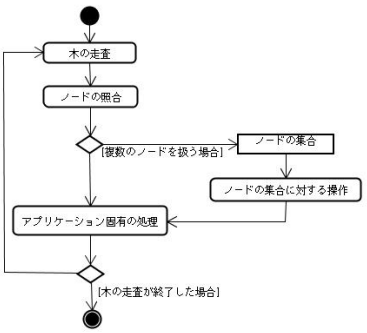

図 2　DOM 木処理の流れ (アクティビティ図)

2.3　記述の問題点と再利用や修正

図 1 の XML 文書を扱う DOM プログラムの例を図 3 に示す．このプログラムは雇用者名の一覧を表示する．木の走査を「再帰呼び出し」の手法を用いて処理を行い，switch 文によってノードの型の照合処理をおこなっている．図 3 のような DOM 木を扱った XML 文書処理系

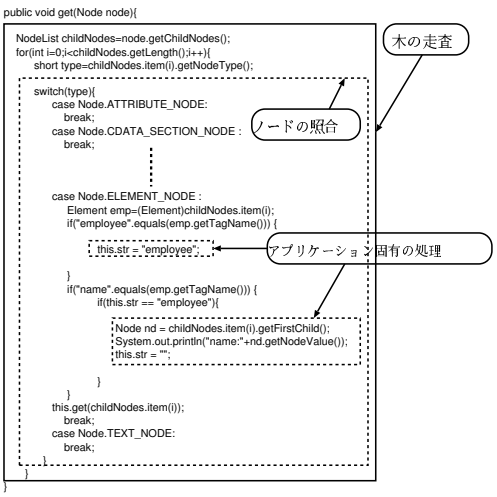

図 3　DOM プログラムの例

におけるプログラムでは，木の走査とノードの照合，アプリケーション固有の処理が混在している．このことから，2.2 節で示した DOM 木の各処理が明確に分離できていない．

XML 文書処理系特有の変更点として，DTD のデータ構造定義の一部変更や異なる XML ボキャブラリへの変更，同じ XML ボキャブラリを利用した異なるアプリケーションへの変更などが挙げられる．これらの変更点に対して，プログラムの修正や再利用が考えられるが，DOM 木の各処理が明確に分離できていないことからプログラムの再利用や保守が困難である．DOM 木処理における再利用性，保守性を考慮したアーキテクチャを構築することで，この問題を解決できることが考えられる．

3 再利用性・保守性を考慮した DOM 木処理におけるデザインパターンの適用

2.3 節で示した問題点を解決するために，再利用性，保守性の向上を目的とし，DOM 木の処理にデザインパターンを適用する．2.2 節の各処理に対して考える．

3.1 品質要求

DOM 木の処理における再利用性，保守性以外の品質要求として，変更性，使用性，効率性が挙げられる．再利用性，保守性以外のこれらの品質特性も含めて考慮し，DOM 木の各処理にデザインパターンを適用する．

効率性は DOM 木の処理にデザインパターンを適用したことによって，適用しない場合と比べて実行効率がどう変わるのかという点で評価を行う．しかし，デザインパターンを適用しないときは直接メソッドを実行していたが，デザインパターンを適用すると，一度別のメソッドを実行してからメソッドを呼び出すなどの処理が増加するので，効率性に関する実行効率は低下する．

3.2 木の走査

木の走査には Interpreter パターンを適用することが考えられる．この場合，DOM 木を構成する各 Node クラスに Interpret() メソッドを定義することで木の走査を行う．しかし，既存の DOM パーサではあらかじめ決められた Node クラスのインスタンスによって構成される DOM 木を生成するので，Interpret() メソッドが定義された DOM 木を生成することができない．このことから，Interpreter パターンを木の走査に適用することができない．

3.2.1 内部 Iterator を利用した木の走査

Iterator パターンを利用して DOM 木の走査を行う場合，DOM 木のような再帰的な集約構造においては内部 Iterator を利用できる．内部 Iterator を利用した DOM 木の走査を行うクラス図を図 4 に示す．DOMDriver クラスに実行すべきオペレーションが定義された Visitor を渡すことで，DOMDriver クラスが木の走査，ノードの照合を行いながら，その Visitor を DOM 木の各ノードに対して適用する．

内部 Iterator は木の走査を行うオブジェクトから他の処理を行うオブジェクトを呼び出し処理を行うので，木の走査をカプセル化する．2.3 節で挙げた変更に対して修正を行う場合は，ノード毎に呼び出すオブジェクトを修正する．開発者は，木の走査を「再帰呼び出し」の手法を用いて DOMDriver クラスに記述する．また，ノード毎にオブジェクトを呼び出すロジックを記述する．

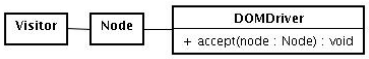

図 4　内部 Iterator

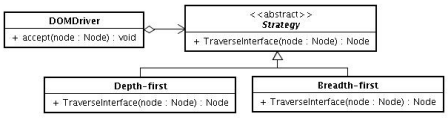

図 5　Strategy パターン

3.2.2 Strategy パターンを利用した木の走査

木の走査には複数の探索方法が考えられる．Strategy パターンでは木の探索アルゴリズムの集合を定義し，各アルゴリズムをカプセル化して，それらを交換可能にする．Strategy パターンを利用した DOM 木の処理を行うクラス図を図 5 に示す．Strategy クラスにアルゴリズムに共通のインタフェースを宣言し，サブクラスで探索アルゴリズムの実装を行う．TraverseInterface() メソッドはノードを引数として，次のノードを返す処理を行う．探索アルゴリズムを実装し共通のインタフェースで実装した探索アルゴリズムを選択することで再利用が可能となる．探索アルゴリズムを実装するので 2.3 節で挙げた変更に対して修正を行う必要はない．開発者は探索アルゴリズムの実装を行う．また，探索アルゴリズム毎に Strategy クラスのサブクラスを作成して実装を行う．

3.3 ノードの照合処理

Visitor パターンを利用することで 2.2 節で述べたノードの照合処理を行う．図 6 に Visitor パターンを利用した DOM 木の処理のクラス図を示す．

XML では要素名によってデータの意味付けを行うので，要素名毎にアプリケーション固有の処理が記述されたメソッドを用意する．ノードの照合処理によってそのメソッドを呼び出し処理を行う．ノードの照合処理を行うメソッド，アプリケーション固有の処理に関する記述

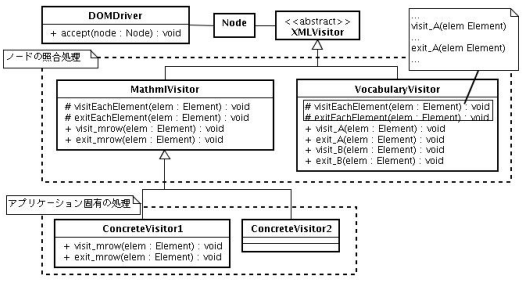

図 6　Visitor パターン，TemplateMethod パターン

を別のオブジェクト (Visitor クラス) にまとめ，木を走査するとき各ノードにこのオブジェクトを渡す．XML ボキャブラリに依存しないノード型に対する処理を行うメソッドを Visitor クラスの XMLVisitor クラスに定義する．XMLVisitor クラスのサブクラスに XML ボキャブラリに対応した VocabularyVisitor クラスを定義する．VocabularyVisitor クラスには要素名毎の処理を行うメソッド，要素名の照合処理を行うメソッドを定義する．DOM 木を走査するプログラムから Visitor クラスのメソッドを呼び出すことでノードの照合を行う．

Visitor パターンを利用することで，木の走査，ノードの照合，アプリケーション固有の処理を分離することができる．また，XML ボキャブラリ毎に依存したノードの照合処理を行うことができる．2.3 節で挙げた変更に対して修正を行う場合は，Visitor クラスを修正する．開発者は，XML キャブラリ毎にノードの照合行うメソッドと要素名毎のメソッドを用意した XMLVisitor クラスのサブクラスを作成する．

3.4 アプリケーション固有の処理

Command パターンを利用することで，複数のノードに対する処理をまとめて扱うことができる．しかし，XML を利用した DOM 木処理では要素名毎で扱う処理が異なると考えられるので Visitor，Template Method パターンを適用する．

DOM 木の処理の 2.2 節で述べたアプリケーション固有の処理に Template Method パターンを適用する．図 6 にクラス図を示す．

Visitor パターンと Template Method パターンを組み合わせることで，XML ボキャブラリ毎にアプリケーション固有の処理の記述ができる．XMLVisitor のサブクラス (VocabularyVisitor クラス) で定義したメソッドを利用し，異なるアプリケーション毎にそのサブクラス (ConcreteVisitor クラス) を作成し，要素名毎に対応したメソッドを実装する．DTD のデータ構造定義の一部変更に対しても, 同様にサブクラスを作成し，対応する照合処理のメソッド内部や要素名毎に対応したメソッド部分を変更する．

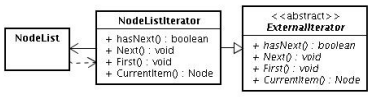

図 7　外部 Iterator

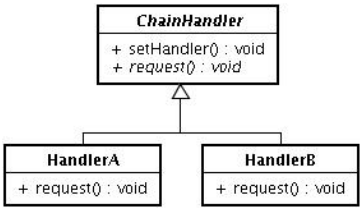

図 8　Chain of Responsibility パターン

Template Method パターンを利用することで，同じ XML ボキャブラリで異なるアプリケーション固有の処理を扱うことが可能となる．2.3 節で挙げた変更に対して修正を行う場合は，VocabularyVisitor クラスのサブクラスを修正する．開発者は VocabularyVisitor クラスのサブクラスを作成し, 要素名毎のメソッドを実装する．

3.5 ノードの集合に対する操作におけるデザインパターンの利用

3.5.1 外部 Iterator

DOM 木の処理においてアプリケーション固有の処理を行うために，木の走査を行ったさいに特定のノードをリスト等のデータの集合としてノードを集約する場合が考えられる．図 7 の外部 Iterator のクラス図では NodeList の各要素にアクセスする方法を提供している．ExternalIterator クラスに要素にアクセス，操作を行うためのインタフェースを定義し，そのサブクラスで ExternalIterator クラスで定義したインターフェースの実装を行う．例に示した NodeList 以外の操作を行う場合は，ExternalIterator クラスのサブクラスで定義し，必要な場合はサブクラスにオペレーションを追加する．

外部 Iterator を利用することで，ノードの集合とその操作に関する記述を明確に分けることができる．また，様々な集約したノード毎のアクセス，操作方法の再利用が可能となる．集約したノードに依存するので，2.3 節で挙げた変更に対して修正を行う必要はない．開発者は，ExternalIterator クラスのサブクラスを作成し，そのサブクラスで定義したメソッドを利用する．

3.5.2 Chain of Responsibility パターン

XML 文書の構造定義によって，DOM 木のノード間には親子関係，兄弟関係などの関係性がある．これらの関係を順に走査していき，適切なオブジェクトで処理を行うことが考えられる．この場合に Chain of Responsibility パターンを用いる．図 8 のクラス図では参照したノードから特定のノードをたどっていき動的に要求を渡していき処理を行う．様々なデータ構造，要求毎に ChainHandler クラスのサブクラスを実装する．

集約オブジェクトと行いたい処理を分離することができる．また，様々な処理を扱い，再利用が可能となる．ノード間の関係性に依存した処理を行う ChainHandler クラスのサブクラスを作成するので，2.3 節で挙げた変更に対して修正を行う場合は，そのサブクラスを修正する．開発者は，ChainHandler クラスを作成する．

3.6 デザインパターンを組み合わせた DOM 木の処理

3 節で述べた各デザインパターンを組み合わせた全体のクラス図を図 9 に示す．アプリケーションが要求する処理によって，DOM 木の処理に適用したデザインパターンの組み合せが考えられる．例として，要素名毎に対して処理が必要な場合に有効な組み合せには，図 9 で示した内部 Iterator，Strategy，Visitor，TemplateMethod パターンの利用が考えられる．他の例として，木の走査で取得したノードから特定のノードの関係を調べて処理を行う場合に有効な組み合わせには，内部 Iterator，

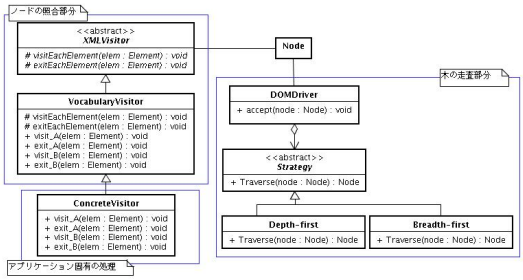

図 9　全体のクラス図

Strategy，Chain of Responsibility パターンの利用が考えられる．また，木の走査においてノードを集約し，そのデータを用いて処理を行う場合に有効な組み合わせには，内部 Iterator，外部 Iterator，Strategy パターンの利用が考えられる．

4　考察

4.1　デザインパターンの組み合わせに関する考察

3.6 節で述べた内部 Iterator，Strategy，Visitor，TemplateMethod パターンの組み合わせは，各デザインパターンを利用することで 2.2 節で述べた DOM 木の各処理を明確に分離できる．Visitor,Template Method パターンにより異なる XML ボキャブラリやアプリケーション固有の処理を扱い，保守性，使用性，変更性を満たす．また，処理の例として，ある Element ノードの属性の id を参照している idref を持つノードを集約し，そのノードの集合を操作し Text ノードなどを扱うことが挙げられる．この場合，内部 Iterator，Strategy, 外部 Iterator パターンの組み合わせを利用することが考えられる．木の走査 (Strategy) とノードの集合に対する操作方法 (外部 Iterator) を再利用し，異なるアプリケーションに対してノードの集合を取る操作 (内部 Iterator) とアプリケーション固有の処理を変更箇所として明確にすることで保守性，使用性，変更性を満たす．

4.2　再利用性を考慮したプログラムコードの記述に関する考察

DOM 木処理のプログラムコードは，デザインパターンの適用により一部形式的に記述されることや，アプリケーションや DOM 木のデータ構造に依存しない部分の処理の記述がある．内部 Iterator，Strategy パターンを適用することで開発者が作成する DOMDriver クラス，Strategy クラス，などは XML ボキャブラリやアプリケーションに依存しないので作成しておけばこれの再利用できる．Visitor パターンによる記述は，XML ボキャブラリ毎に用意した要素名毎の処理を行うメソッドの定義やそれらを呼び出す要素名の照合処理を記述した VocabularyVisitor クラスの再利用が可能になる．我々は MathML 数式を TEX 数式とまた HTML 数式に変換するプログラムを作成し，この再利用性を確認した．VocabularyVisitor クラスは，大規模な XML ボキャブラリを扱う場合などは記述量が膨大になり記述に手間がかかるという問題点が挙げられるが，スキーマ言語を利用して Visitor クラスを自動生成する方法が考えられる．

4.3　デザインパターンを用いた処理の一般化に関する考察

本研究では XML 文書を木構造のデータとして表現する DOM 木を利用した処理のみにデザインパターンを適用した．DOM 木は Node クラスのサブクラスのインスタンスで構成されており，XML では Element ノードの要素名などを利用して処理を行っている．同じように，抽象構文木でもノードが構文要素の種類によってクラス，サブクラスで表現され，そのノードを利用して処理を行う．このことから，DOM 木と同一の表現方法による木構造のデータを利用した処理において 3 節で述べた各デザインパターンを適用できることが考えられる．例えば，2.3 節で XML 文書処理特有の変更点として述べた DTD のデータ構造定義の一部変更，XML ボキャブラリなどの変更に対して行われる修正や再利用方法は，抽象構文木では文法の変更に対応することができると考えられる．しかし，4.1 節で述べた IDREF 型を用いた処理において，抽象構文木では表現方法が異なりデザインパターンの適用は困難であると考えられる．

4.4　関連研究

文献 [1] は，解析した DTD から要素名に対応したメソッドや照合処理を行うメソッドを出力し，DOM 木を処理するための Visitor クラス生成系を提案している．文献 [1] では Visitor パターンで DOM 木を処理する方法を示しているが，その構造や再利用などに関する整理や，Visitor パターンを用いない DOM 木の処理については示されていない．本研究では，Visitor パターン以外のデザインパターンについても DOM 木処理に適用し，Visitor パターンを含めてそれぞれ処理を整理してアーキテクチャを提案した．また，Visitor パターンを用いない DOM 木の処理についても示した．

5　まとめ

本研究では DOM 木の一般的な処理について整理し，DOM 木の処理における再利用性，保守性を考慮したデザインパターンを適用した．また，品質特性の各観点から各デザインパターンの評価をおこない，DOM 木を用いた XML 文書処理系のアーキテクチャを提示した．
今後の課題として，Visitor パターンを用いた記述をスキーマ言語などから自動生成することが挙げられる．

参考文献

[1] 蜂巣吉成,“名前空間を考慮した DOM 木 Visitor 生成系,” ソフトウェア工学の基礎 X, Nov. 2003, pp. 161-164.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, オブジェクト指向における再利用のためのデザインパターン, 1999, ソフトバンク出版.