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 XML 文書木の生成–

2004MT002 朝比奈 和哉　　2004MT010 深谷 整
指導教員　蜂巣 吉成

1　はじめに

XML の普及に伴い，XML 文書を扱うアプリケーションが増加している．XML 文書を利用するアプリケーションは DOM パーサにより DOM 木を構築して処理を行うことが多い．DOM パーサを用いることで, アプリケーションは XML 文書解析の手間を削減することができる．
DOM 木はアプリケーションに依存しない汎用的な中間形であり, アプリケーションにとって DOM 木は必ずしも扱いやすい形式とは言えない．DOM 木を用いたアプリケーション開発では，アプリケーションが行いたい処理に加え，要素の照合やテキストクラスのオブジェクトから文字データへの変換を行うなど，DOM 木に対する処理も行わなければならない．
アプリケーションに適した固有の XML 文書木（以下，固有木）を作成することでアプリケーション開発の手間を削減することができる．しかし，開発者はアプリケーション毎にパーサを作り替えることになり，手間がかかる．
本研究では，解決方法としてデザインパターンを用いた XML パーサのアーキテクチャを提案し，作成手順と再利用箇所を明確にすることを目的とする．これにより，開発者はアプリケーションに特化した固有木を作成するための固有のパーサ（以下，固有パーサ）を容易に作成することが出来る．また，品質特性の観点から固有パーサの評価を行う．
研究手順を次に示す．
・DOM パーサである Xerces の調査
・デザインパターンを用いたアプリケーション固有のパーサを構築するアーキテクチャの提案
・固有パーサの変更点の明確化
・デザインパターンを品質特性の観点から評価

2　DOM パーサとその問題点

2.1　DOM パーサの概要

DOM パーサは XML 文書を読み込むと，文書を構成している要素や属性，テキストデータなどを DOM 木と呼ばれる木構造にしてメモリに展開する [1]．図 1 に XML 文書と DOM 木の例を示す．図 1 は，名前，生年月日，住所を表す XML 文書を DOM 木に変換した例である．

2.2　DOM 木における問題点

DOM 木はアプリケーションに依存しない木構造であり，アプリケーション開発者が構造などを変更することはできない．例えば，ノードには子ノードを取得するなどの基本的なメソッドが実現されているが，新たなメソッドを追加することはできない．DOM 木では XML 文書の要素は全て Element クラスのオブジェクトであり，文字データ（PCDATA）は全てテキストクラスのオブジェクトとなっている．例えば，アプリケーションが図 1 の DOM 木を用いて誕生日を数値として取得するには，birthday 要素の子要素の day 要素を取り出し，その子要素のテキストデータを取り出し数値に変換する必要があり，記述が繁雑になる．

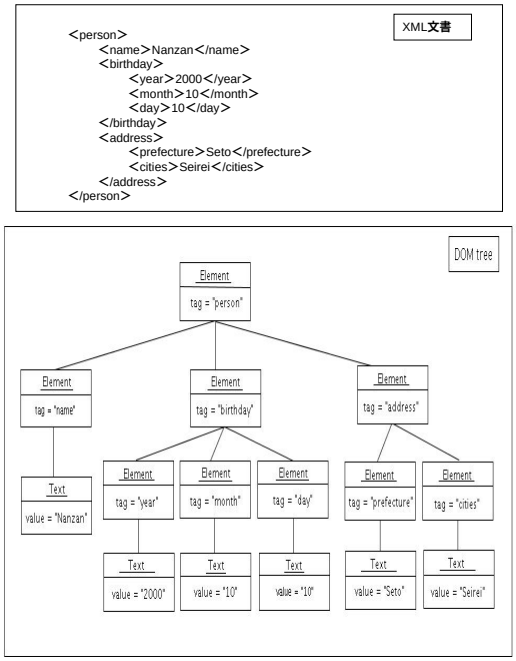

図 1　XML 文書と DOM 木

2.3　Xerces の概要

本研究では XML パーサのアーキテクチャを提案する．そこで，既存の XML パーサである Xerces[2] の調査を行った．既存のパーサを調べることで，パーサの構造を再利用可能な部分と新たに作成する部分に整理し，それをもとに新しいパーサを作ることを考えたからである．
Xerces が XML 文書から DOM 木を作成するまでの流れを図 2 に示す．
Xerces では, スキャナが XML 文書を読み込み，ドキュメント情報を作成し検証系に渡す. 検証系がドキュメント情報の検証を行いパーサに渡す．パーサが渡された情報をもとに木の構築を行うという処理を行っている．
Xerces のスキャナと検証系はアプリケーションによら

ず共通の処理であり，アプリケーションが直接利用するのは，パーサの部分だけである．そこで，スキャナと検証系を再利用をし，検証系から渡された処理により木を作るパーサの部分をアプリケーションに特化したパーサに作り替えることを提案する．

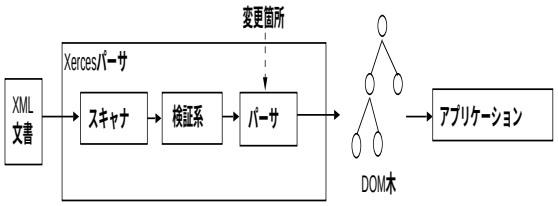

図 2　Xerces の概要

3　固有パーサアーキテクチャへのデザインパターンの適用

3.1　本研究のアプローチ

図 3 の（a）が DOM パーサを用いたアプリケーション開発の流れ，（b）が我々の研究で扱う固有パーサを用いたアプリケーション開発の流れである．DOM パーサを用いたアプリケーション開発では，アプリケーションが行いたい処理に加え，DOM 木に対する処理も行わなければならず，手間がかかっていた．

それに対し，我々の研究では，開発者が固有パーサと固有木を構成するクラスを作成する．2.2 節で挙げたようなアプリケーションが行っていた DOM 木に対する処理をパーサ部分で行うことで，アプリケーション開発の労力を減らす．さらに，固有パーサの開発の手間が増えないようにパーサのアーキテクチャを提案する．

本節では，開発手順が明確になるように，固有パーサが作成する固有木を定義し，パーサの処理を分類してそれぞれの処理にデザインパターンの適用を行う．

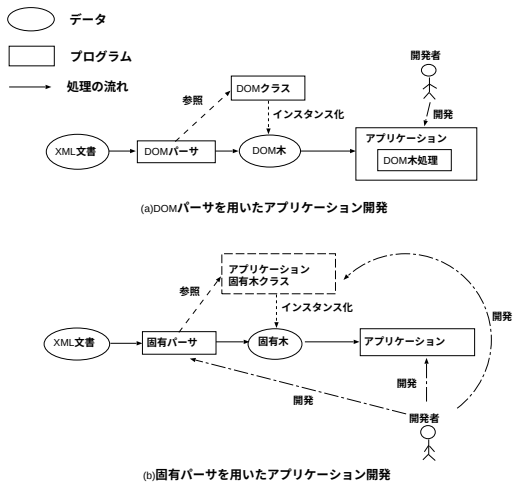

図 3　パーサによる処理の流れとパーサを利用したアプリケーション開発

3.2　アプリケーション固有木

XML 文書は次の二種類に分類できる．アプリケーション固有木もそれに応じて分類できると考える．

・データ中心の XML 文書

　要素の子は複数の要素か 1 つのテキストなので，木の構造が単純になる (図 1).

・ドキュメント中心の XML 文書

　要素の子に複数の要素とテキストが混在しているので，テキストノードと要素ノードを同じものとしてパーサを考え，それに合わせてアプリケーション固有木を作らなければならない．(図 4)

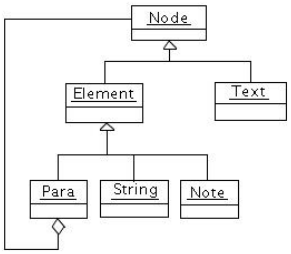

図 4　ドキュメント中心の XML 文書

図 5 にアプリケーション固有木クラスの例を挙げる．アプリケーション固有木を作成することで，2.2 節で挙げた問題点に対し，木のクラスに処理を定義することでアプリケーションは Birthday クラスの getday を呼び出すだけで処理を行うことができる．さらに，現在の年齢を知りたいときには，現在の日時と誕生日から年齢を算出するメソッドをクラスに追加することもできる（図 5 の getAge).

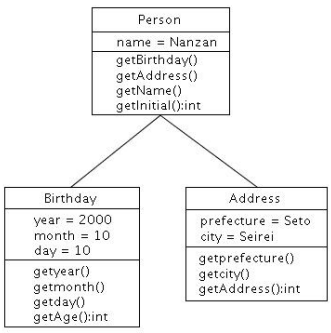

図 5　アプリケーション固有木のクラス

3.3　デザインパターンを用いたパーサの構築

我々は，アプリケーションに特化した木を作成するためには DOM パーサでは不十分であり，新たなパーサを作る必要があると考えた．しかし，アプリケーションが変わる度に，開発者が新たにパーサを始めから構築しなければならず，手間がかかるという問題点が出てきてしまう．

そこで本研究では，デザインパターンを用いることで，パーサを毎回作り替えることなく再利用できるパーサのアーキテクチャを提案する．パーサのアーキテクチャを示すことで作成手順と再利用箇所が明確になる．
デザインパターンは「生成に関するパターン」,「構造に関するパターン」,「振る舞いに関するパターン」の 3 つに分けることができる [3]．本研究では，アプリケーション固有の木を生成するのに，生成に関するパターンを用いる．
固有のパーサを作るにあたり，パーサの変更にはどのような場合があるかを示す．
・固有木のノードにメソッドやサブクラスが追加された場合
・固有木の構造が変更された場合
・ボキャブラリ（XML 文書を記述するための要素や属性）が変更された場合
これらの変更から固有パーサの処理はノードの作成と木の構築の 2 つの処理に分類できる．ノードにメソッドやサブクラスが追加された場合は，木のノードを替え，固有木の構造に変更があった場合は，木の構築を替えることで対応する．ボキャブラリに変更があった場合には，木の構築とノードの生成をする前のボキャブラリの定義を行うクラスに変更を加えることで対応する．

3.3.1 ノードの作成

アプリケーションごとにアプリケーション固有木のクラスが変わることがあり，開発者はクラスが変わる度にパーサを作り替えなければならない．
そこで，メソッドやサブクラスがノードに追加された場合に，処理の追加やオブジェクトの交換が容易になるように，ノード作成の部分で AbstractFactory パターンを適用する．
図 6 に AbstractFactory パターンを用いたクラス図を示す．四角に囲まれた部分の PersonWithInitial などのサブクラスがアプリケーション固有木を定義するクラスである．ファクトリクラスがこれらの固有木を定義しているクラスから木のノードとなるインスタンスを作成している．また，アプリケーション固有木を変更する際には，ファクトリクラスを使い分けることで変更を行うことができる．
さらに，アプリケーションに変更があった際には，開発者はクラスのインスタンスを作成するファクトリクラスに変更を加えることでアプリケーション固有木を作成することができる．

3.3.2 木の構築

一連のオブジェクトを作成し固有木を作るには，パーサではノードを関連付けて木を構築する必要がある．
メソッドやサブクラスの追加がされる場合には，3.3.1 節で提案した AbstractFactory を用いてノードを作成することでパーサの変更に対応させた．木の構造が変更される場合には，木の構築が容易になるように，Builder の適用をする．
図 7 の Builder のクラス図では，抽象クラスである

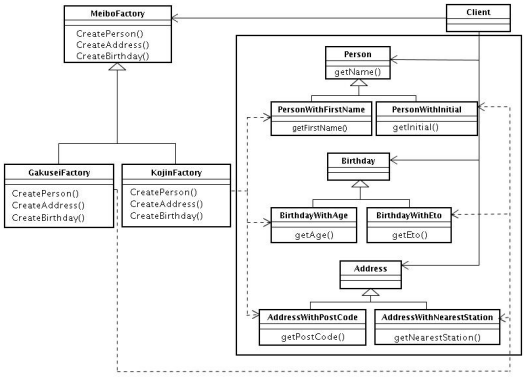

図 6　Abstract Factory クラス

Builder クラスがアプリケーション固有木を作るためのインタフェースを提供している．
また，MeiboBuilder クラスでは，3.3.1 節で述べた AbstractFactory を利用してノードを作り，親子関係の設定を行い，木の構築を行っている．

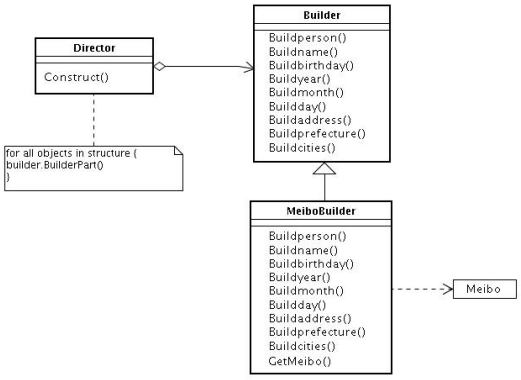

図 7　Builder クラス

3.3.3 アーキテクチャ

図 8 は Builder と AbstractFactory を組み合わせたクラス図である．Xerces から渡されたドキュメント情報を基に，XMLBuilder が要素や属性の並び方を記述したスキーマを定義し，XMLDispatcherBuilder が渡されたスキーマから要素の照合 Text から文字データへの変換を行っている．生成された要素，文字データから MeiboFactory がノードの作成を行い，MeiboBuilder が生成されたノードの親子関係の設定を行い，木の構築を行っている．

4 考察

4.1 固有木の変更に関する考察

3.3 節で挙げた変更点について，図 8 から固有パーサの変更方法を挙げる．
固有木のノードにメソッド，サブクラスの追加があった

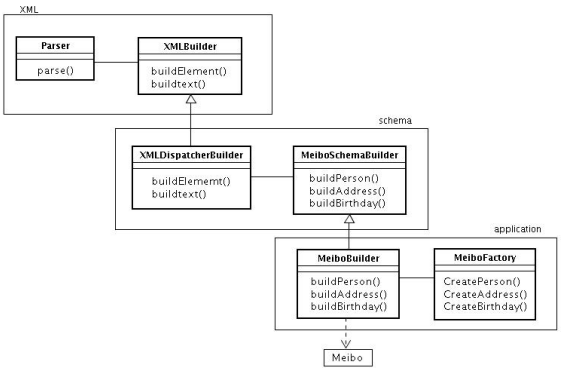

図 8　アーキテクチャの全体像

際には，MeiboFactory の固有木を定義するサブクラスに変更を加え，ノードのインスタンスを新たに作成することで対応ができると考える．
固有木の構造が変更された場合には，ノードから親子関係の設定と木の構築を行う MeiboBuilder について変更を行うことで新たな固有木を構築できると考える．
ボキャブラリが変更された場合には，Element，Text の照合を行う MeiboSchemaBuilder と MeiboBuilder と MeiboFactory それぞれに変更を加えて対応する．
これらの考察から，固有パーサの変更方法と再利用箇所が明確になり，パーサ開発の労力が削減できると考える．

4.2　品質特性の観点からの評価

品質特性のなかで，保守性と効率性から評価を行う．デザインパターンを用いない固有パーサでは，アプリケーションや XML 文書の変更に応じてメソッドやサブクラスをパーサに追加，変更をすることはできるが，新たにパーサを作り替える必要が出てくる．
それに対し，デザインパターンを用いたアプリケーション開発では，4.1 節で挙げた考察から，保守性の副特性である変更性が高まると考える．しかし，ボキャブラリが変更された際には，Element，Text の照合，ノードの作成，木の構築を行う部分に変更を加えなければならず，変更性が高まらなかった．
また，ドキュメント中心の XML 文書に出現する改行タグや段落のタグなど，文書から同一データの想定をし，Flyweight パターンを用いて対象となるインスタンスを生成し，保存しておくことで，ノードを作成する際に同一のデータが出現する場合には，保存しておいた対象のインスタンスを返すことで，メモリを削減することができると考える．

4.3　関連研究

本研究で提案した固有パーサと JAXB[4] の比較を行った．JAXB では，定義した XMLSchema から JAVA の API を生成し，生成された API を通じて木を作成して XML データにアクセスすることで，データを取得している．このことから，JAXB はパーサと固有木の自動生成を行っているので，開発の手間を省くことができる．
しかし，メソッドの追加や木構造の変更ができないことから，XML 文書を扱うのに JAXB では不十分ではないかと考える．本研究では，毎回作り替えることなく再利用できるパーサのアーキテクチャを提案し，再利用箇所を示した．このことから，パーサのカスタマイズ範囲が広がり，JAXB よりも柔軟性が高くなったと考える．

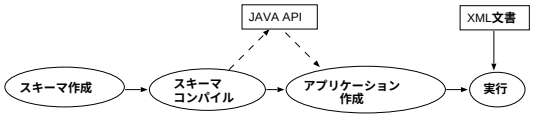

図 9　JAXB 処理の流れ

5　おわりに

本研究では，アプリケーションが行っていた要素の照合，テキストクラスのオブジェクトから文字データに変換するなどの DOM 木に対する処理をパーサ部分で行い，アプリケーションの作業を減らすことを考えた固有パーサの提案をした．
3 節では，ノードの作成と木の構築について，デザインパターンを用いてアーキテクチャの提案をし，再利用箇所を示した．4.1 節では固有パーサの変更方法を明確にした．これにより，パーサの柔軟性が高くなり，固有パーサ開発の労力を減らすことができたと考える．さらに，品質特性の観点から固有パーサの変更性について評価した．
今後の課題として，提案したアーキテクチャに基づいてアプリケーション開発をし，DOM パーサを扱う場合に比べてどれだけ効率が良くなるか評価することが挙げられる．

謝辞

本研究を進めるにあたり，御指導いただいた野呂昌満教授，沢田篤史教授，蜂巣吉成准教授，御世話になった大学院生のみなさまに深く感謝いたします．

参考文献

[1] J. Graf, P. Houle, A. K. Hughes, D. Hollander, J. Duckett, K. K. Hughes, F. Boumphrey, P. Jones, C. McQueen,and O. Direnzo, *XML APPLICATIONS*, SHOEISHA, 2000.
[2] The Apache Software Foundation, *Xerces Native Interface*, 1999; http://xerces.apache.org/xerces2-j/xni.html.
[3] E. Gamma, R. Helm, R. Johnson,and J. Vlissides, *Elements of Reusable Object-Oriented Software*, SOFT BANK PUBLISHING, 1995.
[4] Sun Developer Network (SDN), *Java Architecture for XMLBinding (JAXB)*, Mar. 2003; http://java.sun.com/developer/technicalArticles/WebServices/jaxb/.