ORB ミドルウェアのためのアスペクト指向ソフトウェア・アーキテクチャの提案

蜂巣吉成[†]　野呂昌満[†]　張漢明[†]

本研究では ORB ミドルウェアの構造を整理し開発を省力化するために，ORB ミドルウェアのアスペクト指向ソフトウェア・アーキテクチャを提案する．ORB ミドルウェアを階層モデルで表現すると複数の層を横断した構成要素が存在してしまう．われわれは層をアスペクトとしてとらえてアスペクト間の関係を整理した．アーキテクチャに基づいて，ORB のフレームワークとスタブ，スケルトンの自動生成系を設計・実現する．ORB に関するアスペクトは独立したフレームワークとして実現する．自動生成系はウィーバの役割をし，ORB フレームワークを利用するためのコードが織り込まれたスタブ，スケルトンを生成する．提案したアーキテクチャに基づいて XML-RPC の ORB ミドルウェアを実現し，開発が省力化できることを確認した．

Aspect-Oriented Software Architecture for ORB Middleware

Yoshinari HACHISU,[†] Masami Noro[†]
and Han-Myung CHANG [†]

We propose an aspect-oriented architecture for ORB middlewares. ORB middlewares sometimes are considered as layered model, but a layer has relationships with several layers. We recognize a layer as an aspect, and organize an architecture. Based on our architecture, we design frameworks for ORB aspects and stub/skeleton generators. Frameworks designate spots to be developed. Generators weave aspects into stub or skeleton code. We implemented an ORB middleware for XML-RPC, and we confirmed effectiveness of our approach.

1. はじめに

近年，分散オブジェクト技術を用いたアプリケーション開発が一般化している．代表的な分散オブジェクト技術として CORBA[1)]，Java RMI[2)]，XML-RPC[3)]，SOAP[4)] などが提案され，ORB ミドルウェア☆が開発されている．

しかし，ORB ミドルウェアの構造の整理が不十分であり，開発が個々の開発者の知識や経験に依存するという問題がある．ORB ミドルウェアの構造を階層モデルにより表現することがあるが，技術により各層の位置が異なる場合がある．技術に依存しない一般化した階層モデルを考えると，階層を横断した構成要素が存在し，階層モデルは適切ではない．

本研究では ORB ミドルウェアの構造を整理し開発を省力化するために，ORB ミドルウェアのアスペクト指向ソフトウェア・アーキテクチャを提案する．アスペクト指向では，あるアスペクトを実現するコードが複数のアスペクトに散在するときに，それらをひとまとめにして分離することができる．アスペクト指向の概念を用いることで ORB ミドルウェアにおける複数の層を横断した構成要素を整理することができる．

アーキテクチャに基づいて，ORB のフレームワークとスタブ，スケルトンの自動生成系を設計・実現する．ORB に関するアスペクトは独立したフレームワークとして実現する．自動生成系はアスペクトを結合するウィーバの役割をし，ORB フレームワークを利用するためのコードが織り込まれたスタブ，スケルトンを生成する．

XML-RPC の ORB ミドルウェアをフレームワークと自動生成系を利用して実現した．フレームワークにより可変部をあつらえることで ORB を開発でき，自動生成系により ORB フレームワークを利用するためのスタブ，スケルトンのコードをサーバ・オブジェクトから生成することができ，開発が省力化されるこ

† 南山大学 数理情報学部
Faculty of Mathematical Sciences and Information Engineering, Nanzan University

☆ 本論文では ORB とスタブ，スケルトンのことを ORB ミドルウェアと呼ぶ (図 1).

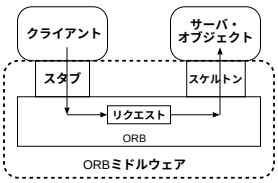

図 1 ORB ミドルウェア

表 1 分散オブジェクト技術の性質と交換形式

言語独立性	特定のプログラミング言語に依存せずに ORB ミドルウェアを利用できる性質
位置透過性	分散アプリケーションの開発者が，サーバオブジェクトの位置を意識せずに利用できる性質．スタブ，スケルトンが，サーバオブジェクトの位置決めやメソッド呼出しの中継を行うことで位置透過性が実現される
整列形式	CPU などに依存しないデータの表現形式
通信規約	メッセージを送受信するための通信規約

とを確認した．

2. 分散オブジェクト技術

表 1 に示す分散オブジェクト技術が考慮すべき性質や，ORB を特徴づけるメッセージ交換形式の観点から，CORBA, Java RMI, XML-RPC, SOAP を概観し，階層モデルとしてとらえた場合の問題点を指摘する．

2.1 CORBA

CORBA(Common Object Request Broker Architecture) は OMG によって提案された規格である [1]．言語独立性，位置透過性を満たすためにサーバ・オブジェクトのインタフェースは IDL と呼ばれる言語で記述し，IDL コンパイラによって Java や C++用のスタブ，スケルトンのコードが生成される．

整列形式は CDR(Common Data Representaion) と呼ばれるバイト順の選択が可能なバイナリ表現が用いられる．TCP/IP での通信規約には，IP アドレスやポート番号，CDR のバイト順などが格納さた IIOP(Internet Inter-ORB Protocol) が用いられる．

2.2 Java RMI

Java RMI(Remote Method Invocation) は Sun Microsystems によって提案された位置透過性を満たした分散オブジェクト技術である [2]．Java 言語での利用を前提としており，言語独立性は考慮されてない．サーバ・オブジェクトは Java 言語で記述され，JavaRMI コンパイラによって Java 用のスタブ，スケルトンのコードが生成される．

データの整列化には Java 言語仕様として定義されたオブジェクト直列化 (Serialization) が用いられる．通信には JRMP と呼ばれる TCP/IP を利用した Java RMI 用の通信規約，または HTTP が用いられる．

2.3 XML-RPC, SOAP

XML-RPC と SOAP は言語独立性を考慮して，整列形式に XML，通信規約に HTTP を用いた遠隔手続き呼び出しの規約である [3][4]．XML-RPC, SOAP はともに整列形式に XML を用いるが，XML によって定義される語彙は異なる．また，スタブ，スケルトンの利用は規定されておらず，位置透過性は考慮されていない．

CORBA や Java RMI が通信規約と整列形式，ORB の実行時環境，スタブ，スケルトンまでを定義しているのに対し，XML-RPC と SOAP は通信規約と整列形式を規定しているだけであり，厳密には ORB ミドルウェアとは言えない．しかし，近年注目されている Web サービスの根幹をなす技術であるので，ORB ミドルウェアとして実現することは重要である．

2.4 階層モデルにおける問題点

Java RMI は，スタブ/スケルトン層，リモート参照層，トランスポート層の 3 つの層からなる階層モデルとして実現されている．スタブ/スケルトン層は分散アプリケーションと ORB の間でデータの橋渡しをおこなう．リモート参照層はサーバ・オブジェクトの管理を，トランスポート層はデータ通信処理をおこなう．

このモデルではデータの整列化，非整列化はスタブ/スケルトン層でおこなわれる．しかし，整列化と非整列化は CORBA，Java RMI，XML-RPC などの各技術に依存した処理なので，データの橋渡し処理とは分離して新たな階層 (整列層) にすべきである．XML-RPC と SOAP は通信層と整列層のみを定義した規約であるので，この分離は妥当である．

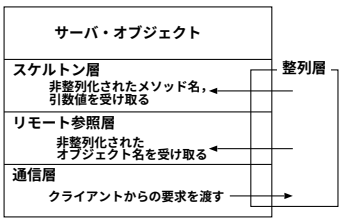

図 2 整列層と他の層との関係

階層モデルにおける整列層の位置を決定するために，ここではサーバ側における整列層と他の層との関係を考える．通信層では受信したクライアントからの要求を整列層に渡す．リモート参照層では整列層から非整列化されたサーバ・オブジェクトの名前を受け取

り，要求されているサーバ・オブジェクトを検索する．スケルトン層では整列層から非整列化されたメソッド名や引数の値を受け取る．したがって，整列層は通信層，リモート参照，スケルトン層を横断して関係を持ち (図 2)，階層構造として表現できない．

3. ORB ミドルウェアのアスペクト指向ソフトウェア・アーキテクチャ

本研究では ORB ミドルウェアにおいて複数の層を横断する整列化/非整列化処理をアスペクト指向の概念を用いて整理する．

アスペクト指向では，ある特性や機能により規定されたコンサーンによって定義されたソフトウェアの一部分をアスペクトとする．アスペクトは合流点 (Join Point) で他のアスペクトと結合する．本論文では，アスペクト指向ソフトウェア・アーキテクチャを，ソフトウェアをアスペクトで表現し，合流点におけるアスペクトの関係を示したものとして定義する．

階層モデルでは ORB ミドルウェアのクライアントとサーバを同じ階層構造としてとらえていたが，本来クライアントとサーバではコンサーンが異なるので独立して考える．

3.1 クライアント

独立した処理という観点からクライアントのコンサーンを表 2 のように認識し，従来の階層モデルにおける各層をアスペクトとしてとらえた．

クライアント側ではスタブがサーバ・オブジェクトと同じメソッドをアプリケーションに提供し，スタブのメソッドを呼び出すと要求がサーバに中継される．メソッドの中継処理の実現を考えると，引数の整列化，サーバ・オブジェクト名の整列化，サーバ・オブジェクトへのデータ通信，返り値の非整列化の処理がスタブの各メソッドに散在するので，これらをアスペクトとして整理する．

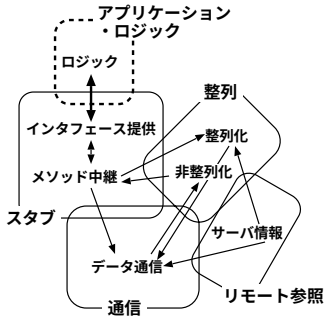

図 3 クライアントのアスペクト間の関係

図 3 にアスペクト間の合流点を表現したクライアントのアスペクトの関係を示す．われわれは階層モデルにとらわれず，アスペクト間の制御とデータの流れからアスペクト間の関係を決定した．図の四角はアスペクト，アスペクト内のテキストはそのアスペクトを規定するコンサーンを表す．アスペクトが重なっている部分が合流点であり，矢印はアスペクト間の制御またはデータの流れを表す．表 3 には代表的な合流点と，合流点における処理を示す．

3.2 サ ー バ

クライアントと同様にしてサーバのソフトウェア・アーキテクチャを構築した．表 4 にアスペクトとコンサーンを，図 4 にアスペクトの関係を，表 5 に代表的な合流点と合流点における処理を示す．

サーバ側ではスケルトンがクライアントからの要求をサーバ・オブジェクトのメソッド呼出しに振り分ける．メソッドの振分け処理の実現を考えると，受信した要求の非整列化，返り値の整列化の処理がスケルトンのコードに散在するので，これらをアスペクトとして整理する．

表 2 クライアントのアスペクトとコンサーン

コンサーン	説明	アスペクト
データ通信	HTTP などの通信規約に従ってデータを送受信する	通信
整列化	データを整列化する	整列
非整列化	整列化されたデータを計算機の内部表現に復元する	
サーバ情報	サーバ・オブジェクトの位置情報を管理する	リモート参照
インタフェース提供	クライアント側の ORB 利用者に対して，サーバ・オブジェクトと同一のインタフェースを提供する	スタブ
メソッド中継	ローカルのメソッド呼出しを遠隔メソッド呼出しに変換する	

表 3 クライアントの合流点における処理

合流点	合流点における処理
ロジック → インタフェース提供	サーバ・オブジェクトのメソッドを呼び出す
メソッド中継 → 整列化	メソッド名，引数を渡して整列化する
サーバ情報 → 整列化	サーバ・オブジェクト名を渡して整列化する
整列化 → データ通信	整列化されたデータを渡す
サーバ情報 → データ通信	通信先の情報を渡す
メソッド中継 → データ通信	遠隔メソッド呼び出しの要求をサーバに送信する

表 4 サーバのアスペクトとコンサーン

コンサーン	説明	アスペクト
待受け通信	クライアントからの接続を待つ	通信
データ通信	通信規約に従ってデータを送受信する	
整列化	データを整列化する	整列
非整列化	データを非整列化する	
レジストリ	サーバ・オブジェクト名とスケルトンの対応を管理する	リモート参照
メソッド振分け	クライアント側の要求をサーバ・オブジェクトに振り分けてメソッドを実行する	スケルトン

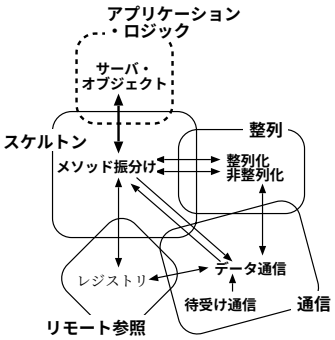

図 4 サーバのアスペクト間の関係

表 5 サーバの合流点における処理

合流点	合流点における処理
データ通信 ↔ 非整列化	クライアントの要求を渡して非整列化されたサーバ・オブジェクト名を受け取る
データ通信 ↔ レジストリ	サーバ・オブジェクト名を渡して対応するスケルトンを受け取る
データ通信 → メソッド振分け	クライアントの要求をスケルトンに渡す
メソッド振分け ↔ 非整列化	メソッド名，引数のデータを渡して非整列化されたデータを受け取る
メソッド振分け ↔ サーバ・オブジェクト	クライアントから要求されたサーバ・オブジェクトのメソッドを呼び出して，返り値を受け取る

4. アーキテクチャに基づいた ORB ミドルウェアの設計と実現

3 節で提案したアスペクト指向ソフトウェア・アーキテクチャにしたがって，ORB ミドルウェアを設計・実現する．

スタブ・アスペクトとスケルトン・アスペクトの実現には自動生成系を用いる．スタブはサーバ・オブジェクトと同じインタフェースをもち，メソッド呼出しをサーバに中継する．スタブは分散アプリケーションに依存したメソッドを持つので，コンポーネントやフレームワークとしてインタフェースを標準化することはできない．一方，スタブのメソッドは決まった手順で整列化や通信などの処理をおこなうので，処理内容を定型化することは可能である．以上のことからサーバ・オブジェクトのインタフェースからスタブを自動生成する方法を選択した．スケルトンも同様である．

ORB を実現するリモート参照，通信，整列アスペクトは，コンポーネントとして実現する方法とフレームワークとして実現する方法が考えられる．われわれは，フレームワークよる ORB の実現を選択した．各アスペクトは独立したフレームワークとして実現し，スタブ，スケルトンの自動生成系がアスペクト間の関係を生成する．フレームワークにより，インタフェースを標準化し，分散オブジェクト技術に依存した開発箇所を局所化かつ明確化することができる．コンポーネントによる実現では，コンポーネントを組み合わせてソフトウェアを構築できるが，ORB では技術固有の箇所が多く，コンポーネントの組み合わせはほとんど起らないので，利点が得られない．

4.1 ウィーバと自動生成系

一般にアスペクト指向言語では，アスペクトと結合処理は独立して記述され，ウィーバ (weaver) と呼ばれる処理系がアスペクトが結合されたコードを生成する (図 5)．アスペクトは既存のプログラミング言語で，結合処理は専用のプログラミング言語で記述されることが多い．例えば，Java 言語と結合処理記述言語の AspectJ[5] などがある．

ORB ミドルウェアでは結合処理として，アスペクトの順序を指定して起動することや，起動したアスペクト間でのデータの受渡しを記述する必要がある．例えば，スタブでは整列 (整列化)，通信，整列 (非整列化) のような順序でアスペクトを起動してメソッドの中継を行う．また，各アスペクトはメソッドの実引数や返り値によりデータの受渡しを行う．しかし，一般の結合処理記述言語およびウィーバではこれらは考慮されていない．

本研究では，位置透過性を満足した ORB ミドルウェアにはスタブ，スケルトンの自動生成系が存在することに注目し，自動生成系をウィーバとみなしてアスペクトを結合する方法を提案する．スタブ，スケルトンはサーバ・オブジェクトのインタフェースから生成され，ORB を利用してメソッドの中継やメソッドの振分けをおこなう処理が記述される．これは，スタブ・アスペクトとスケルトン・アスペクトにデータ通信アスペクトや整列アスペクトが結合されたと考えられる．応用領域を ORB ミドルウェアに限定することで，汎

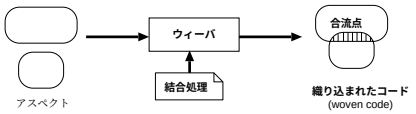

図 5 ウィーバ

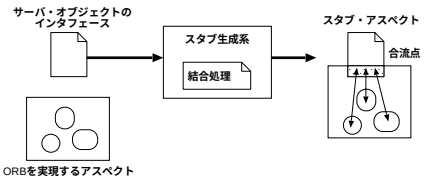

図 6 自動生成系

用のウィーバを用いた場合では考慮されていなかったアスペクトの起動順序の指定やアスペクト間のデータの受渡し処理を定型化して記述し，自動生成することができる．また，自動生成されるコードをアスペクト指向により整理することで，自動生成系の構造を整理することができる．

図 6 に自動生成系の処理の概略を示す．自動生成系はアスペクトの結合処理を内包したウィーバであると言える．

4.2 自動生成系の設計と実現

自動生成系の入力であるサーバ・オブジェクトのインタフェースの例を図 7 に，生成するスタブ・コードの例を図 8 に示す☆．例には Java 言語を用いた．

スタブ・アスペクトのインタフェース提供コンサーンは図 7 で定義されるメソッド・インタフェースとして表現され，メソッド中継コンサーンはメソッドの本体として表現される．スタブの各メソッドに散在する整列化処理，通信処理，非整列化処理は自動生成系のアスペクトの結合処理にしたがって生成される．つまり，自動生成系は結合処理にしたがって，メソッド中継コンサーンに整列アスペクト (図 8 の 14〜16 行) や通信アスペクト (図 8 の 20 行) などを結合したコードを生成する．

```
1: public interface Sample {
2:   int compare(int a, int b);
3:   int[] sort(int ar[]);
4: }
```

図 7 サーバ・オブジェクトのインタフェース

われわれは以下の指針にしたがって自動生成系を設計する．

(1) 自動生成系を言語から独立して設計・実現するためにデザイン・パターン [6] を用いる

(2) コード生成処理を，コードの記述言語に依存する部分と独立した部分に分離する．

(3) 各合流点における結合処理を個別に記述する．

ここではスタブの自動生成系について説明するが，スケルトンの自動生成系も同様にして設計・実現できる．

☆ スケルトンのコードは紙面の都合で省略した．

```
 1: public class SampleStub implements Sample {
 2:   private ServerInfo sinfo;
 3:   public SampleStub(ServerInfo sinfo) {
 4:     this.sinfo = info;
 5:   }
 6:   public int compare(int param_0, int param_1) {
 7:     /*** メソッド名、引数の整列化処理 ***/
 8:     ClientMarshalStream ms
 9:       = new XmlrpcClientMarshalStream();
10:     ms.open();
11:     // 整列とリモート参照の合流点
12:     ms.writeObjectName(sinfo.getObjectName());
13:     // 整列とスタブの合流点
14:     ms.writeMethodName("compare");
15:     ms.writeInt(param_0);
16:     ms.writeInt(param_1);
17:     ms.close();
18:     /*** 通信処理 ***/
19:     // スタブと通信の合流点
20:     Connection con = new HttpConnection();
21:     // 通信とリモート参照の合流点
22:     con.connect(sinfo);
23:     // 整列と通信の合流点
24:     con.sendRequest(ms.getBytes());
25:     byte res[] = con.recieveResponce();
26:     con.disconnect();
27:     /*** 返り値の非整列化処理 ***/
28:     ClientUnmarshalStream ums
29:       = new XmlrpcClientUnmarshalStream(res);
30:     ums.open();
31:     // 整列とスタブの合流点
32:     int result = ums.readInt();
33:     ums.close();
34:     return result;
35:   }
36:   public int[] sort(int[] param_0) {
37:     /*** メソッド名、引数の整列化処理 ***/
38:     ...
39:     /*** 通信処理 ***/
40:     ...
41:     /*** 返り値の非整列化処理 ***/
42:     ...
43:     reeturn result;
44:   }
45: }
```

図 8 スタブのコード

4.2.1 スタブ・コードの構造

自動生成系は特定のプログラミング言語のためのコードを生成する．指針 (2) にしたがって言語独立部分を考えた場合，スタブ・コードの構造が独立部分として挙げられる．図 8 で示したようなスタブ・コードは図 9 のような抽象構文木で表現できる．スタブ・コードではさらに文や式といった構文要素が出現するが，文法を細粒度で表現すると言語への依存度が高く

なり，木も大きくなる．われわれは木の節点として，特定の言語に依存しないクラスやメソッド，および，通信，整列アスペクトの合流点を選んだ．自動生成系は抽象構文木を深さ優先探索して節点に対応するコードを生成するように設計した．

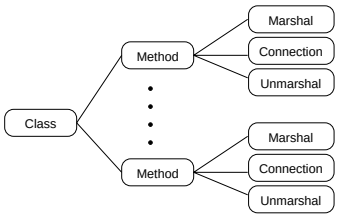

図 9 スタブの抽象構文木

4.2.2 コード生成部の設計

木の探索処理には，抽象構文木を解釈実行するためのパターンである Interpreter パターンを用いる．

木探索時に各節点のコード生成処理を記述するには以下の 3 つの方法が考えられる．

(1) Visitor パターン
(2) Command パターン
(3) サブクラスと Template Method パターン

Visitor パターンは節点に対応した処理を Visitor クラスのメソッドとして記述する．(1) は Interpreter パターンで木を探索し，節点訪問時に Visitor クラスのメソッドを呼び出してコードを生成する．しかし，1 つの Visitor クラスにすべての合流点の結合処理が記述されるので指針 (3) に反する．例えば，HTTP で通信をおこなう XML-RPC と SOAP では通信アスペクトの結合処理記述を再利用したいが，Visitor パターンでは XML-RPC 用の XmlrpcVisitor クラス，SOAP 用の SoapVisitor クラスに個別に記述しなければならない．

Command パターンは要求，すなわちコード生成処理を一つのクラスにカプセル化することができる．(2) では抽象構文木の節点の種類に対応したコード生成処理が記述された Command クラスを定義する．コード生成は，Interpreter パターンで木を探索し，節点の種類に応じた Command オブジェクトを生成し，呼び出すことでおこなわれる．Command オブジェクトの生成は Abstract Factory パターンを用いることでカプセル化できる．

(3) では木の節点クラスに木を探索する処理と抽象メソッドとして定義されたコード生成処理を記述する．実際のコード生成処理はサブクラスでメソッドをオーバライドして記述する (Template Method パターン)．サブクラスのオブジェクトの生成は Abstract Factory パターンを用いることでカプセル化できる．

(2) はコード生成に委譲を用いる方法，(3) は継承を用いる方法と言える．自動生成系ではどちらの方法でも同じように実現できるが，継承はオブジェクト指向の基本概念の一つであると考えて，(3) の方法により自動生成系を設計した．

設計したクラス図を図 10 に示す．ClassNode, MethodNode などが木を探索する処理 (generate) とコード生成のための抽象メソッドが定義されたクラスである．JavaClassNode, JavaMethodNode などが Java 言語用のコード生成処理が記述されたクラスである．合流点における結合処理は JavaMarshalClassNode などのメソッド (gnerateJPRemoteRef) として記述する．アスペクト間の関係は，アスペクトの起動順序とアスペクト間のデータの受渡しとして表現される．起動順序は，メソッドを呼び出す順序として Template Method パターンにより記述される．データの受渡しはあらかじめ決められた名前の変数，例えば図 8 の 25 行目の res を利用して，返り値を受け取り，実引数として渡すコードを生成することで実現する．構文木は Abstract Factory パターンに従った JavaFactory クラスが節点のオブジェクトを生成して構築される．

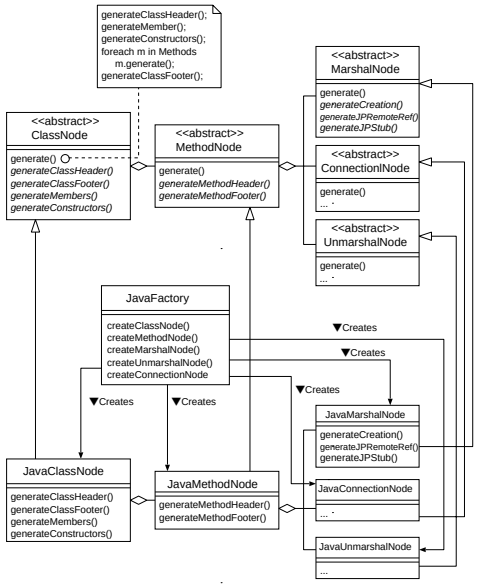

図 10 スタブ生成系のクラス図

4.3 ORB フレームワークの設計と実現

整列アスペクトなどの ORB を実現するアスペクトは技術によって処理内容が大きく異なる場合が多いの

で，ホワイトボックス・フレームワークとして設計・実現する．フレームワークによりインタフェースを統一し，継承を利用して技術に依存しない部分を不可変部，依存する部分を可変部とする．

図 11 にクライアントの通信，リモート参照アスペクトを実現するフレームワークのクラス図を示す☆．

通信アスペクトでは接続 (connect)，切断 (disconnect)，送信 (sendRequest)，受信 (receiveResponce) のための抽象メソッドを定義した Connection クラスが不可変部である．実際の分散オブジェクト技術では通信に TCP を用いることが多いので，TCP による接続処理，切断処理を TcpConnection クラスで定義し，不可変部とした．HTTP などの通信規約にしたがって通信をおこなうクラスは可変部で，送信処理と受信処理を定義する．

フレームワークを分散オブジェクト技術から独立して利用するには，利用者には Connection クラスなどの不可変部のクラスのみを提供し，HttpConnection などの可変部のクラスを隠蔽しなければならない．可変部のオブジェクトの生成には Factory Method パターンや Abstract Factory パターンを用いることが多い．しかし，これらのパターンを用いると独立性が低下する．

われわれが提案する ORB ミドルウェアの実現法ではスタブ，スケルトンの自動生成系が存在するので，可変部オブジェクトの生成処理は自動生成系がスタブ，スケルトンに記述することにした (図 8 の 9, 20, 29 行)．これにより，フレームワークのクラスによる機能の実現とオブジェクトの生成処理を分離することができる．また，アスペクト間の関係，すなわちフレームワーク間のデータの受渡し処理も自動生成系が生成することで (図 8 の 22 行)，フレームワークの独立性を高めることができる．

5. 議　　論

5.1 アスペクト指向に関する議論

本研究では ORB ミドルウェアの構造をアスペクト指向を用いて整理した．従来，階層を横断していた整列化/非整列化処理をアスペクトとしてとらえ，独立したフレームワークで実現することができた．スタブ，スケルトンの自動生成系を応用領域を ORB ミドルウェアに限定したウィーバとして設計することにより，自動生成系の構造を明確にして，アスペクトの起動順序の指定やアスペクト間のデータの受渡し処理を定型化することができた．

☆ 紙面の都合上，整列アスペクトとサーバ・フレームワークのクラス図は省略するが，同様にして設計した．

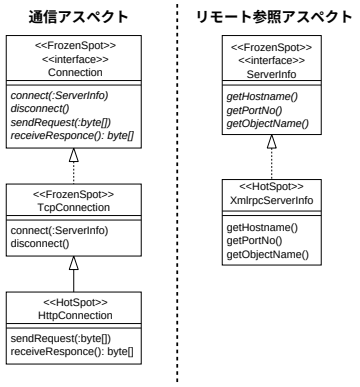

図 11 通信，リモート参照アスペクトのフレームワーク

拡張性に関する議論

アスペクト指向の概念を用いたことにより，ORB の機能を拡張することも容易であると考えられる．既存の ORB フレームワークを変更することなく新しい機能を追加でき，デザインパターンを用いることで自動生成系の変更を局所化することができる．ここでは，ORB に通信データの暗号処理を追加することを例にして拡張性を議論するが，実時間処理などを追加する場合も同様である．

ソフトウェア・アーキテクチャの観点では，暗号化と復号化をコンサーンとして考え，暗号化と復号化によっておこなわれる暗号処理を 1 つのアスペクトとして考える．ORB ではデータ送信前に暗号化，データ受信後に復号化する必要があるので，暗号アスペクトと通信アスペクトは関係を持つ (図 12)．

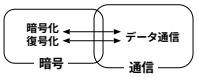

図 12 暗号処理におけるアスペクトの関係

ORB の他のアスペクトと同じように暗号アスペクトは独立したフレームワークとして設計・実現される．暗号アスペクトと通信アスペクトのデータの受渡し処理は自動生成系がスタブ，スケルトンに生成する．したがって通信アスペクトのフレームワークを変更することなく，暗号の機能を追加することができる．

自動生成系では図 10 の JavaConnectionNode にデータの受渡し処理を追加する必要がある．JavaConnectionNode への処理追加は継承や Decorator パターンなどを用いることで，変更箇所を局所化することができる．

5.2 ORB フレームワークの実現に関する議論

Java 言語を用いて ORB フレームワークを実現し，XML-RPC 用の ORB を実現して，以下の観点から ORB ミドルウェア開発が省力化されることを確認した．

- 整列化などの技術に強く依存した処理記述は再利用できないが，その他の処理は再利用可能である．

  ORB フレームワークの不可変部は約 580 行，可変部は約 1380 行である．可変部の通信に関する部分が約 150 行，整列化・非整列化に関する部分が約 1170 行である．通信規約として HTTP を用いた通信部は不可変部としてコード量を計算しているが，SOAP などの他の技術でも再利用可能な部分である．XML-RPC では XML で整列化をおこなうが，XML パーサを利用した処理や int や int の配列などの型に応じた処理記述のためにコード量が多くなっている．
- 開発箇所が明確化される．

  可変部の実現では，データの受渡し処理やオブジェクトの生成処理を考える必要がなく，注目しているアスペクトの機能の実現に専念することができた．また，4.3 節で示したように各アスペクトにおいて実現すべき箇所がクラスやメソッドとして明示されていた．

5.3 自動生成系の実現に関する議論

Java 言語を用いて Java 言語用のスタブ，スケルトンを生成する自動生成系を実現した．サーバ・オブジェクトの構文解析には Java の Reflection API を利用した．自動生成系の規模は約 1210 行である．自動生成系により，図 7 のサーバ・オブジェクトのインタフェースから，60 行のスタブ・コード，56 行のスケルトン・コードが生成される．

自動生成系により，スタブ，スケルトンの開発を省力化でき，位置透過性を満たして XML-RPC を利用することができた．

5.3.1 技術に対する独立性

自動生成系が生成するスタブ，スケルトンのコードで ORB に依存した部分は，フレームワークの可変部オブジェクトを生成する処理である．図 8 では 9, 20, 39 行目が相当する．実現した自動生成系ではこれらのクラス名をコマンドライン引数として与えることにより技術に依存しないようにした．

5.3.2 言語に対する独立性

例えば C++言語のスタブ，スケルトンを自動生成系により生成するには，図 10 の ClassNode, MethodNode などを継承して C++言語のコードを生成するクラス CppClassNode, CppMethodNode などを定義し，それらのインスタンスを生成する CppFactory クラスを定義すれはよい．

自動生成系はデザイン・パターンを利用して設計されているので，自動生成系自体を C++などで実現することも容易である．なお，この場合は抽象構文木の作成には，サーバ・オブジェクトのインタフェースを構文解析する必要がある．

6. おわりに

本論文では ORB ミドルウェアの構造を整理し開発を省力化するためにアスペクト指向ソフトウェア・アーキテクチャを提案した．ORB の各機能，すなわちアスペクトは ORB フレームワークにより独立して実現され，アスペクトの結合はスタブ，スケルトンの自動生成系によっておこなわれる．Java 言語を用いてフレームワークと自動生成系，XML-RPC 用の ORB ミドルウェアを実現し，開発が省力化されることを確認した．

今後の課題としては以下が挙げられる．

- SOAP などの他の分散オブジェクト技術を実現して，フレームワークにおける再利用率などを計測し，提案したアーキテクチャの有効性を示す．
- アスペクト指向を用いて自動生成系を設計・実現する．自動生成系のアスペクトを構文解析，木走査，コード生成などとして構造を整理し，各アスペクトを独立して実現することで，ソースコードの読解性の向上や機能追加に対する変更の容易さが期待できる．

参 考 文 献

1) OMG: *The Common Object Request Broker: Architecture and Specification* (1999).
2) Sun Microsystems: *Java Remote Method Invocation Specification*.
3) UserLand Software: *XML-RPC Home Page*. http://www.xmlrpc.com/.
4) W3C Note: *Simple Object Access Protocol (SOAP) 1.1* (2000). http://www.w3.org/TR/SOAP/.
5) AspectJ Team: *The AspectJ Programming Guide*. http://aspectj.org/.
6) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns*, Addison-Wesley (1994).