SOA アプリケーションプラットフォームのプロダクトライン化に関する研究
—再利用コンポーネントの定義， 設計， 実装—

2009SE124 小林利誌　2010SE244 富塚祐太
指導教員：野呂昌満

1　はじめに

本研究室では，SOA アプリケーションプラットフォーム (以下, App.PF) のプロダクトライン化に関する研究 [2] が行なわれている．App.PF のプロダクトライン化によって，ミドルウェアの選択に影響されない SOA システムの開発が支援される．プロダクトラインアーキテクチャはアスペクト指向に基づいて構築している．アプリケーションフレームワークはプロダクトラインアーキテクチャに基づいて定義している．アプリケーションフレームワークの構築では，ミドルウェアの選択に影響を受けるアスペクトをホットスポットとして定義する．ミドルウェアは製品ごとにインターフェースが異なる．ミドルウェア間の差異を吸収し，ホットスポットにミドルウェアを適合させるための再利用コンポーネント群を用意する．App.PF のプロダクトライン化の課題として，再利用コンポーネントの自動生成が挙げられる．再利用コンポーネントの自動生成を実現するためには，自動生成の仕組みを整理し，自動生成のためのツールを整備する必要がある．

本研究の目的は，再利用コンポーネントの自動生成の仕組みの整理である．自動生成の仕組みを整理することで，App.PF のプロダクトライン構築を支援する．再利用コンポーネントの自動生成の仕組みの整理のために，自動生成可能な箇所を整理し，入力と出力を整理する．

ホットスポットにミドルウェアを適合させるためには，次の 3 つの条件を満たす必要があると考え，その方法について整理した．

- オブジェクトの Semantics が包含関係
- メソッドの Semantics が同一
- メソッドのシグネチャが同一

本研究では，オブジェクトの Semantics は包含関係にあり，メソッドの Semantics は同一であることを前提とする．ミドルウェアの柔軟な変更を実現するために，変動性を実現する仕組み [1] を参考にし，Variation Point(変動点) ごとに統一のインターフェースを定義した．統一のインターフェースとミドルウェアのインターフェースのシグネチャの差異を吸収するために，引数を Semantics で対応づけた．引数の Semantics の対応関係をパターン化することで，引数を対応させるコードが自動生成可能であると考えた．引数を対応させるコードの自動生成の入力と出力を整理した．App.PF のアプリケーションフレームワークのホットスポットで事例検証を行ない，自動生成の仕組みの妥当性を確認した．

2　本研究の前提

再利用コンポーネントは，ホットスポットに適合するミドルウェアの柔軟な変更を実現するためのコンポーネント群である．ミドルウェアの柔軟な変更を実現するために，変動性を実現する仕組みを参考にした．本研究では，Variant ごとにインターフェースが異なるので，Variation Point ごとに統一のインターフェースを定義する．

ホットスポットにミドルウェアを適合させるためには，次の 3 つの条件を満たす必要がある．

- オブジェクトの Semantics が包含関係
- メソッドの Semantics が同一
- メソッドのシグネチャが同一

オブジェクトの Semantics を比較するために，オブジェクトの Operational Semantics を比較する．オブジェクトの Operational Semantics は，状態遷移図で記述し，比較する．ミドルウェアのオブジェクトの状態遷移図が，Variation Point のオブジェクトの状態遷移図を包含していれば，ホットスポットにミドルウェアを適合可能である．本研究では，ミドルウェアのオブジェクトの Operational Semantics は，Variation Point のオブジェクトの Operational Semantics を包含しているものとする．メソッドの Semantics を比較するために，Variation Point のメソッドと，ミドルウェアのメソッドの事前条件，事後条件を比較する．比較するメソッドは，開発者が用意した対応表に基づいて決定する．本研究では，Variation Point のメソッドと，ミドルウェアのメソッドの事前条件，事後条件は包含で，メソッドの Semantics が同一であるとする．これらの前提があるとき，ホットスポットにミドルウェアを適合させるためには，メソッドのシグネチャが同一であればよい．メソッドのシグネチャが同一でなければ，その差異を吸収することで，ホットスポットにミドルウェアを適合させることができる．シグネチャの差異は，統一のインターフェースでミドルウェアのインターフェースをラッピングすることで吸収する．ラッピングするさいに，統一のインターフェースの引数と，ミドルウェアのインターフェースの引数を役割で対応づける必要がある．

3　引数の対応づけ

統一のインターフェースの引数と，ミドルウェアのインターフェースの引数を対応づけるために，引数の Seman-

tics に着目した．引数にそれぞれタグを与え，タグの名前で引数の Semantics を記述した．同一の名前のタグを与えられている引数同士を対応づける．引数に与えたタグが，複数のタグのレコードである場合は，タグにタグパターンを与えることでその情報を記述する．図 1 は，引数に与えたタグのタグパターンを示している．

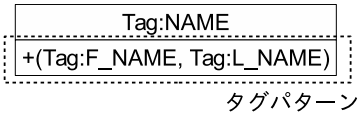

図 1　タグパターン

3.1　タグの対応関係のパターン化

統一のインターフェースの引数に与えたタグと，ミドルウェアのインターフェースの引数に与えたタグの対応関係からパターンを発見し，整理した．引数に与えたタグの対応関係のパターンとして，次の 4 つを定義した．

- 通常対応
- 分割対応
- 結合対応
- 分割 + 結合対応

通常対応は，統一のインターフェースの引数に与えたタグと同一のタグが，ミドルウェアのインターフェースの引数に与えられているパターンである．図 2 では，引数に与えたタグが，通常対応のパターンで対応づいている．分割対応は，統一のインターフェースの引数に与えたタグのフィールドと同一のタグが，ミドルウェアのインターフェースの引数に与えられているパターンである．図 3 では，引数に与えたタグが，分割対応のパターンで対応づいている．結合対応は，統一のインターフェースの引数に与えたタグのレコードと同一のタグが，ミドルウェアのインターフェースの引数に与えられているパターンである．図 2 では，引数に与えたタグが，結合対応のパターンで対応づいている．分割 + 結合対応は，統一のインターフェースの引数に与えたタグのフィールドと同一のタグをフィールドとするタグが，ミドルウェアのインターフェースの引数に与えられているパターンである．図 5 では，引数に与えたタグが，分割 + 結合対応のパターンで対応づいている．

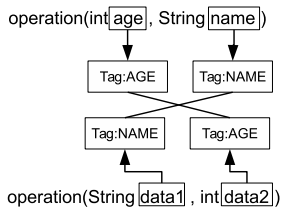

図 2　通常対応

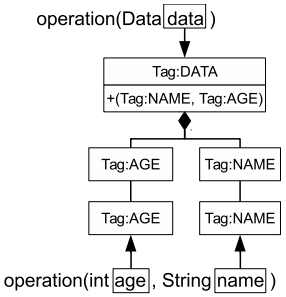

図 3　分割対応

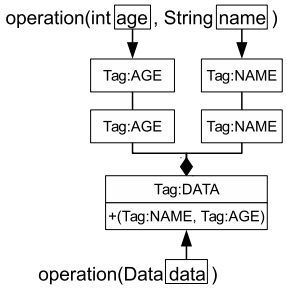

図 4　結合対応

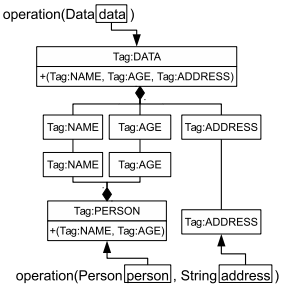

図 5　分割 + 結合対応

4　再利用コンポーネントの自動生成

統一のインターフェースの引数と，ミドルウェアのインターフェースの引数に与えたタグの対応関係をパターン化できたことから，引数を対応させるコードが自動生成可能と考えた．引数と，引数に与えたタグの対応関係を入力とし，引数を対応させるコードを出力とする．通常対応の場合は，入力した統一のインターフェースの引数を，タグが対応しているミドルウェアのインターフェースの引数に渡すコードを出力する．図 6 は，図 2 の通常対応の例で出力する，引数を対応させるコードである．分割対応の場合は，入力した統一のインターフェースの引数を役割で分割し，その役割を持つ値を生成する．生成した値を，タグが対応しているミドルウェアのインターフェースの引数に渡すコードを出力する．図 7 は，図 3 の分割対応の例で出力する，引数を対応させるコードである．結合対応の場合は，入力した統一のインターフェースの引数のレコードをタグの対応関係をもとに生成する．生成したレコードを，タグが対応しているミドルウェアのインターフェースの引数に渡すコードを出力する．図 8 は，図 4 の結合対応の例で出力する，引数を対応させるコードである．分割 + 結合対応の場合は，入力した統一のインターフェースの引数を役割で分割し，その役割を持つ値を生成する．生成した値のレコード，または生成した値と他の引数のレコードを，タグの対応関係をもとに生成する．生成したレコードを，タグが対応しているミドルウェアのインターフェースの引数に渡す値を出力する．図 9 は，図 5 の分割 + 結合対応の例で出力する，引数を対応させるコードである．

```
operation(int age, String name) {
  data2 = age;
  data1 = name;
  middleware.operation(data1, data2);
}
```

図 6　通常対応の場合における引数を対応させるコード

```
operation(Data data) {
  age = data.age;
  name =data.name ;
  middleware.operation(age, name);
}
```

図 7　分割対応の場合における引数を対応させるコード

5　事例検証

App.PF の Variation Point である Message Service Provider(以下，MSP) で，事例検証を行なった．Variation Point MSP は，サービス同士のメッセージングを仲

```
operation(int age, String name) {
  data = (age, name);
  middleware.operation(data);
}
```

図 8　結合対応の場合における引数を対応させるコード

```
operation(Data data) {
  age = data.age;
  name = data.name;
  address = data.address;
  person = (age, name);
  middleware.operation(person, address);
}
```

図 9　分割 + 結合対応の場合における引数を対応させるコード

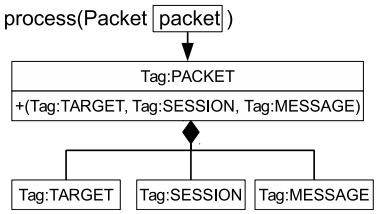

図 10　Variation Point MSP の統一のインターフェースの引数に与えたタグ

介する．Variation Point MSP の統一のインターフェースは, process(Packet packet):void と定義する．引数 packet は，次の 3 つの情報を保持している．

- 自身を送る相手
- App.PF のセッション情報
- 相手に送信するメッセージ

packet を受け取ったサービスは，送られてきたメッセージに応じてオペレーションを実行し，App.PF のセッションを変更する．

Variation Point MSP の Variant であるミドルウェアの選択肢には，次の 3 つが存在する．

- JBossESB
- MuleESB
- ServiceMix

これら 3 つのミドルウェアは, いずれも JMS をサポートしている．よって，これら 3 つのミドルウェアのインターフェースは，process(String name, Message messages):void である．

5.1　引数の対応づけ

統一のインターフェースの引数と，ミドルウェアのインターフェースの引数を Semantics で対応づけるために，引数にタグを与えて Semantics を記述する．統一のインターフェースの引数 packet に Tag:PACKET を与える．引数 packet は，3 つの情報を保持している．それらの情報の Semantics を記述するために，それぞれにタグを与える．自身の送り先の情報に，Tag:TARGET を与える．App.PF のセッション情報に，Tag:SESSION を与える．相手に送信するメッセージの情報に，Tag:MESSAGE を与える．よって，Tag:PACKET は，Tag:TARGET, Tag:SESSION, Tag:MESSAGE のレコードである．Tag:PACKET がレコードであることを示すために，タグパターン +(Tag:TARGET，Tag:SESSION, Tag:MESSAGE) を Tag:PACKET に付加する．図 10 は，統一のインターフェースの引数 packet に与えたタグを示している．

ミドルウェアのインターフェースの引数にタグを与える．引数 name は，メッセージを送る相手の情報を保持している．よって，引数 name には，Tag:TARGET を与える．引数 messages には，Tag:MESSAGES を与える．引数 messages は，App.PF のセッション情報と，相手に送信するメッセージの情報を保持している．よって，Tag:MESSAGES は，Tag:TARGET と Tag:MESSAGE のレコードである．Tag:MESSAGES がレコードであることを示すために，タグパターン +(Tag:SESSION, Tag:MESSAGE) を Tag:MESSAGES を付加する．図 11 は，ミドルウェアのインターフェースの引数 name，messages に与えたタグを示している．

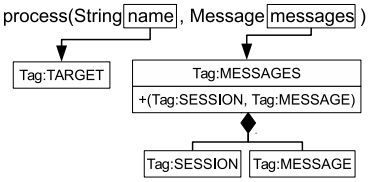

図 11　Variation Point MSP の Variant であるミドルウェアのインターフェースの引数に与えたタグ

統一のインターフェースの引数とミドルウェアのインターフェースの引数に与えたタグの対応関係の中から，定義したパターンを発見する．統一のインターフェースの引数 packet に与えた Tag:PACKET のフィールド Tag:TARGET と同一のタグが，ミドルウェアのインターフェースの引数 name に与えられている．この対応関係は，分割対応のパターンである．統一のインターフェースの引数 packet に与えた Tag:PACKET のフィールド Tag:SESSION，Tag:MESSAGE のレコードである Tag:MESSAGES が，ミドルウェアのインターフェースの引数 messages に与えられている．この対応関係は，分割 + 結合対応のパターンである．図 12 は，Variation Point MSP における，引数に与えたタグの対応関係の中から発見したパターンを示している．統一のインターフェースの引数 packet が持つ 3 つの情報すべてを，ミドルウェアのインターフェースの引数に対応させることができた．よって，統一のインターフェース process(Packet packet):void で，ミドルウェアのインターフェース process(String name，Message messages):void をラッピングすることができる．ラッパーの実現によって，ホットスポットにミドルウェアを適合させることが可能となった．

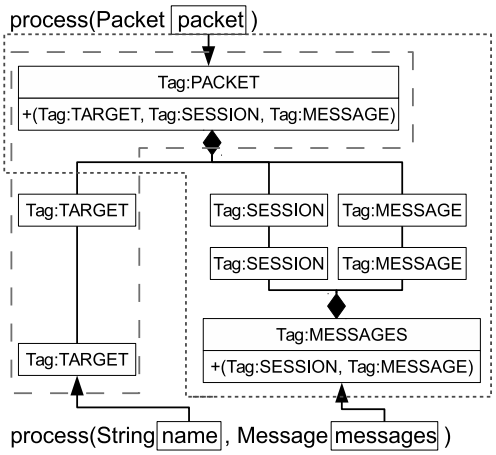

図 12 Variation Point MSP におけるタグの対応関係から発見したパターン

5.2 再利用コンポーネントの自動生成

統一のインターフェースとミドルウェアのインターフェースの引数に与えたタグの対応関係から，定義したパターンを発見した．発見したパターンから，引数を対応させるコードを導出する．引数 packet と引数 name が，Tag:TARGET の通常対応によって対応づいている．よって，引数 packet が持つ，自身を送る相手の情報を，引数 name に渡すコードを自動生成する．引数 packet と引数 messages が，Tag:SESSION，Tag:MESSAGE の分割 + 結合対応によって対応づいている．よって，引数 packet が持つ，セッション情報と，相手に送信するメッセージの情報のレコードを生成するコードを自動生成する．さらに，生成した値を引数 messages に渡すコードを自動生成する．引数と，引数に与えたタグの対応関係のパターンを入力とすることで，引数を対応させるコードを導出することができた．図 13 は，引数と，引数に与えたタグの対応関係のパターンから導出した，出力となる引数を対応させるコードである．

```
process(Packet packet) {
  name = packet.target;
  messages = (packet.session, packet.message);
  middleware.process(name, messages);
}
```

図 13 MSP の事例において入力から導出したコード

6 考察

6.1 引数を対応づける方法の妥当性

統一のインターフェースの引数と，ミドルウェアのインターフェースの引数を対応づける方法について考察する．引数同士を対応づけるために，引数の Semantics に着目した．引数にタグを与え，タグの名前で引数の Semantics を記述した．同一の名前のタグを与えられている引数同士を対応させることで，引数を Semantics で対応させることができると考えた．引数が複数の情報を保持していることを示すために，引数に与えるタグにタグパターンを付加することを提案した．

App.PF の Variation Point である MSP の事例で，タグによって引数を対応づけることが可能か検証した．統一のインターフェースの引数が持つすべての情報を，ミドルウェアのインターフェースの引数に対応させることができた．よって，タグを利用して引数を対応づける方法は妥当であると考える．

6.2 引数を対応させるコードの自動生成の仕組みの妥当性

引数に与えたタグの対応関係のパターンを整理した．引数と，引数に与えたタグ同士の対応関係のパターンを入力することで，引数を対応させるコードが自動生成可能であると考えた．App.PF の Variation Point である MSP の事例で，引数を対応させるコードを導出可能か検証した．引数と，引数に与えたタグの対応関係のパターンを入力とし，引数を対応させるコードを出力とした．統一のインターフェースの引数が持つすべての情報を，ミドルウェアのインターフェースの引数に対応させるコードを導出することができた．よって，提案した自動生成の仕組みは妥当である．

7 おわりに

本研究では，再利用コンポーネントの自動生成の仕組みを整理するために自動生成可能な箇所と入力と出力の整理を行なった．オブジェクトの Semantics が包含関係であり，メソッドの Semantics が同一であるときのメソッドのシグネチャの差異を吸収した．引数を対応させるためにタグを用いて引数の Semantics を記述した．タグの名前から引数を Semantics で対応させた．タグの対応関係をパターン化することで，引数を対応させるコードを自動生成可能であると特定した．自動生成の入力を引数と，引数に与えたタグの対応関係のパターンとし，引数を対応させるコードを出力とした．

本研究の成果は，ある前提のもとで再利用コンポーネントの自動生成の仕組みを整理したことである．今後の課題は，自動生成のためのツールを整備することである．オブジェクトの Semantics が包含関係にないときの再利用コンポーネントの自動生成の仕組みも整理する必要がある．

参考文献

[1] K. Czarnecki, and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[2] 江坂篤侍，野呂昌満，沢田篤史，“ SOA に基づくシステムのためのアプリケーションプラットフォームのプロダクトライン化に関する研究, ” 情報処理学会研究報告．ソフトウェア工学研究会報告，vol. 2013-SE-179, no. 25, pp. 1-6, 2013.