インタラクティブソフトウェアのためのメタ生成系に関する研究

M2015SE004 松下竜巳
指導教員：沢田篤史

1 はじめに

スマートデバイスや Web ブラウザの多様化に伴い，インタラクティブソフトウェアの実行時環境及び開発環境は多様化している．本研究室では，多様な開発環境と実行時環境すべての可能な組み合わせに対する単一モデル開発を目的として，インタラクティブシステムのための共通アーキテクチャ[4]（以降，共通アーキテクチャと呼ぶ）を提案している．共通アーキテクチャの構成要素として，イベント変換系，ビューモデル変換系，イベント処理系が定義されている．イベント変換系は，外部イベントを内部イベントに変換する．ビューモデル変換系は，ビューの内部表現を特定の外部表現形式へ変換する．イベント処理系は，イベントに応じたアプリケーションロジックを実行する．このうち，イベント変換系とビューモデル変換系は入力となるデータ構造を異なる構造および内容のデータ構造に変換する．イベント処理系は，内部に状態遷移機械を保持し，入力となるイベントに応じたアクションを起動する．

アプリケーションに対する要求や使用する開発環境によって，これら変換系，処理系（以降，データ構造変換系と呼ぶ）に入出力されるデータ構造は異なる．アプリケーションの開発支援の方法の 1 つとして，データ構造変換系のソースコードの自動生成が考えられる．

本研究の目的は，共通アーキテクチャのデータ構造変換系を生成する生成系の実現である．この生成系はデータ構造変換系の振る舞いを生成することから，メタ生成系と呼ぶ．メタ生成系の振舞いを柔軟に変更可能とすることで，多様な開発環境と実行時環境の組み合わせに対する開発を容易にする．

メタ生成系はデータ構造変換系とみなすことができると考え，データ構造変換系およびメタ生成系に共通のアーキテクチャを設計する．メタ生成系の振舞いを変更可能とすることを目的とし，データ構造変換系の固定部分と変動部分を明確に分離して定義することを設計指針とする．共通のアーキテクチャは，モデル駆動アーキテクチャ（以降，MDA と呼ぶ）[3] を参考にし，MappingFunction と MappingRule からなるアーキテクチャとして設計する．データ構造変換系の MappingFunction と MappingRule をモジュール化することにより，メタ生成系を用いてそれぞれを独立して生成可能となる．MappingFunction は入力となるプラットフォーム独立モデル（以降，PIM と呼ぶ）に固定される部分であり，出力となるプラットフォームに依存したモデル（以降，PSM と呼ぶ）へ変換する MappingRule は Platform によって変動する部分である．設計したアーキテクチャに基づく実現として，入力となるデータ構造を分類し，標準的な走査手続きを定義する．メタ生成系は，データ構造の種類および変換規則を入力としてデータ構造変換系の振る舞いを生成するものとして実現する．設計したアーキテクチャの妥当性，関連研究 [1] との比較によるアーキテクチャの有用性，データ構造変換系の走査手続きと変換規則を生成可能であることを考察した．

2 背景技術

2.1 共通アーキテクチャ

共通アーキテクチャは，多様化するインタラクティブシステムの実行時環境や開発環境を統一的に扱うことを目的として提案されている．MVC アーキテクチャとその派生の既存のアーキテクチャが分離を試みている横断的関心事を特定し，アスペクト指向アーキテクチャとして共通参照アーキテクチャを設計しており，横断的関心事を選択することで共通アプリケーションアーキテクチャを生成する（図 1）．

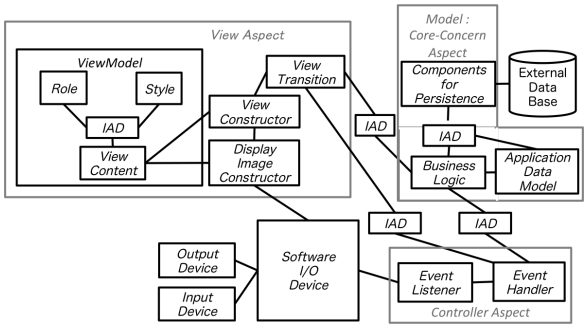

図 1 共通アプリケーションアーキテクチャ

2.2 MDA

MDA は PIM にプラットフォーム情報を付加し，PSM へ変換するためのシステムのアーキテクチャである．

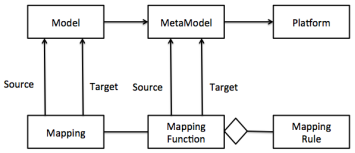

図 2 MDA に基づくモデル変換のアーキテクチャ[3]

図 2 の構成要素について説明する．

- Model：データ変換系の入出力となるモデル
- MetaModel：モデルを定義するためのメタモデル
- Mapping：PIM を受け付け PSM を出力する
- MappingFunction：PIM を走査し MappingRule を適用する
- MappingRule：PIM にプラットフォーム情報を付加し PSM に対応づける変換規則

3　データ構造変換系

データ構造変換系を，出力するデータ構造から独立するために，データ構造を中心とする開発を実現する MDA に基づいて設計する．図 2 をデータ構造変換系に適用すると構成要素は以下のように説明できる．

- Model：データ構造
- MetaModel：データ構造を定義するメタモデル
- Mapping：データ構造の入出力処理
- MappingFunction：データ構造を走査し，MappingRule を適用
- MappingRule：データ構造の要素から異なるデータ構造の要素へ変換

以降に MappingFunction，MappingRule の詳細について説明する．

3.1　MappingFunction

MappingFunction は，入力されるデータ構造を走査し，データ構造の要素に対して MappingRule を適用する．データ構造変換系に入出力されるデータ構造はアプリケーションや実行時環境および開発環境毎に異なる．走査手続きはデータ構造によって固定されると考え，データ構造変換系の入力となるデータ構造を分類する．データ構造変換系の入力となるデータ構造を分類するために，次に分類される共通アーキテクチャのコンポーネントの振る舞いについて整理する．

- イベント変換系は，外部イベント名，外部イベント型のインスタンスを入力とし，内部イベント型のインスタンスを出力する．
- ビューモデル変換系は，ビューの内部表現形式を入力とし，外部表現形式を表すソースコードを出力する．
- イベント処理系は内部に状態遷移機械を保持し，入力となるイベントに応じたアクションを起動する．

以上からデータ構造変換系に入力されるデータ構造は，トークン，センテンス，リスト，グラフ，ツリーのいずれかであると分類する．

分類したデータ構造毎に走査手続きを定義することにより，データ構造変換系の走査手続きを標準化する．入力の形式毎の走査手続きを表 1 にまとめる．

表 1　データ構造毎の走査手続

入力分類	走査手順	補足
トークン	一文字ずつ取得	コンパイラの字句解析器
センテンス	一文字ずつ取得	トークン列を生成
	根から子要素を取得し，再帰呼び出し	解析木を生成
	子要素を取得し行きがけ順に走査	
ツリー	子要素を取得し行きがけ順に走査	
グラフ	子要素を取得し行きがけ順に走査	既走チェックを追加
リスト	子要素を取得し，再帰呼び出し	

3.2　MappingRule

MappingRule は，データ構造の要素を異なる構造及び内容を変換する．データ構造の要素を変換する際には，入力されるデータ構造の要素毎に要素を評価し，適切な変換規則を適用する（図 3）．

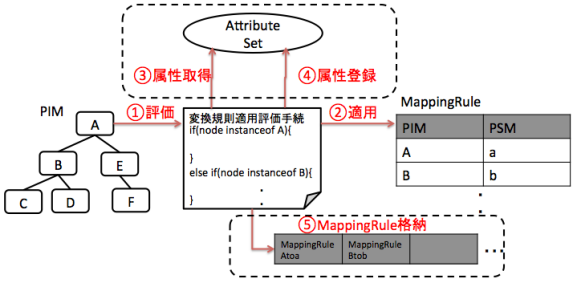

図 3　PIM から PSM への変換手続き

変換規則の適用には，子要素の属性や MappingRule の適用の結果をもとに変換規則を適用する場合や，自身より前に評価された結果や前の要素の属性を引き継ぎ変換規則を適用する場合がある．変換規則の適用時にこの適用順序を表すために，属性評価器を参考にする．図 4 に属性評価器を参考にした MappingRule の構造を示す．

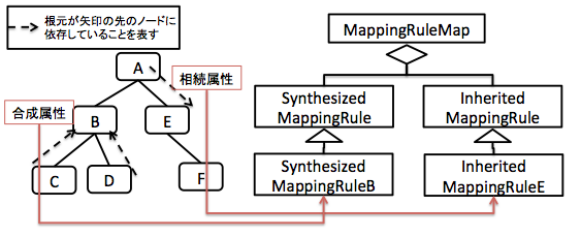

図 4　MappingRule の構造

MappingRule を相続属性用の MappingRule と合成属性用の MappingRule に分離し設計する．

MappingRule の適用順序を示すために，図 3 の破線部で囲われた振る舞いを追加する．AttributeSet は，走査した要素の情報の集合であり，必要に応じて AttributeSet に要素の情報を格納する．MappingRule の評価に必要な情報を AttributeSet から取得する．MappingRule をキューに格納することで一列化し，先頭から順に評価する．

4　メタ生成系

アプリケーションの要求や実行時環境，開発環境に応じたデータ構造変換系を用いた開発を支援するために，データ構造変換系の振る舞いを生成するメタ生成系を設計する．メタ生成系は，モデル中心の開発を実現するためにデータ構造変換系と同様に MDA に基づいて設計する．MDA に基づいて設計したメタ生成系のアーキテクチャは，図 2 に示したデータ構造変換系のアーキテクチャと同一であるデータ構造変換系とメタ生成系の関係を図 5 に示す．

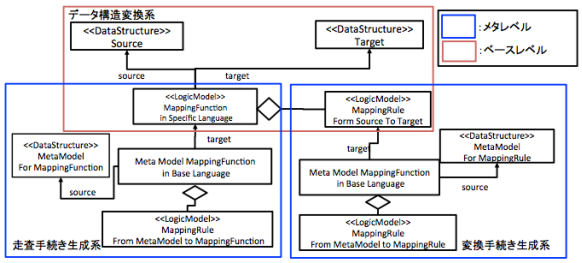

図 5　メタ生成系とデータ構造変換系

メタ生成系は，データ構造変換系の走査手続きを生成する生成系（以降，走査手続き生成系と呼ぶ）とデータ構造変換系のデータ構造の要素を異なる要素，構造のデータ構造へ変換する手続きを生成する生成系（以降，変換手続き生成系）である．

4.1　走査手続き生成系

走査手続き生成系は，データ構造変換系の入力となるデータ構造の定義を入力とし，データ構造変換系の走査手続きを出力する．入力はデータ構造であることから，走査手続きは整理したデータ構造毎の走査手続きを適用する．入力となるモデルのメタモデルを MOF[2] を参考に定義する（図 6）．

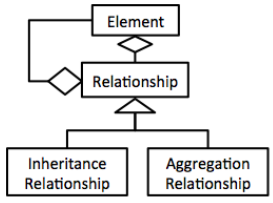

図 6　データ構造のメタモデル

4.2　変換手続き生成系

変換手続き生成系は，データ構造変換系の変換規則を生成する生成系（以降，変換規則生成系と呼ぶ）と変換規則を適用する手続きを生成する生成系（以降，変換規則適用手続き生成系）からなる（図 7）．

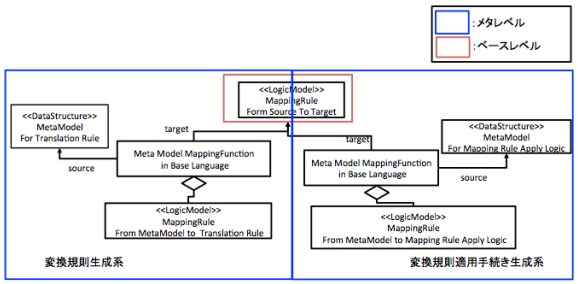

図 7　変換手続き生成系のアーキテクチャ

変換規則適用手続き生成系は，前節で説明した変換規則適用評価処理を生成し，変換規則生成系は MappingRule を生成する．変換規則適用手続き生成系も，データ構造の種類と意味規則を入力とし，変換規則適用評価処理を生成することから，データ構造変換系とみなす．

5　考察

5.1　メタ生成系の妥当性の確認

データ構造変換系のアーキテクチャは，固定部分と変動部分について独立して定義することを設計指針として定義した．MappingFunction はデータ構造毎に固定される部分，MappingRule は実行時環境毎に変化する変動する部分としてモジュール化した．これにより，メタ生成系はそれぞれについて個別に生成可能となった．MappingFunction と MappingRule の組みによって，様々な実行時環境に応じた共通アーキテクチャのデータ構造変換系の振る舞いが生成可能となったと考える．

本稿で提案したデータ構造変換系は，インタラクティブシステムのための共通アーキテクチャを対象とし設計した．データ構造変換系のアーキテクチャは，ドメインを共通アーキテクチャに限定し設計を行った．共通アーキテクチャの各コンポーエントの振る舞いから，入力となるデータ構造を分類し，それぞれの走査手続を標準化した．インタラクティブシステム以外にも，このメタ生成系を適用しようとした場合，データ構造変換系の入力となるデータ構造は変化すると考える．入力となるデータ構造が変化した場合，新たなデータ構造の走査手続きを定義し，データ構造の定義を入力とすることで，このメタ生成系はインタラクティブシステム以外にも応用可能ではないかと考える．

5.2　クロスプラットフォーム開発環境との比較

クロスプラットフォーム開発環境 [1] は，複数の実行時環境上で動作するソフトウェアのワンソースによる開発を可能としている．開発に用いるプログラミング言語は指定され，対象となる実行時環境には制限がある．実行時環境の仕様が変更された場合には，この環境を用いた開発はできない．

一方で，提案するメタ生成系は，プログラミング言語は指定されるが，変換規則の変更のみでデータ構造変換系の実行時環境の追加が可能となる．アプリケーションの実行時環境によって，データ構造変換系の入出力となるデータ構造の組み合わせは変化するが，メタ生成系を用いることで全てのデータ構造の組み合わせ毎にデータ構造変換系を実現する必要がなくなる．実行時環境の仕様が変更された場合，メタ生成系を用いてデータ構造変換系を生成することにより，データ構造変換系の改版が容易になる．

5.3　メタ生成系の実現に関する考察

メタ生成系は，走査手続き生成系，変換規則生成系，変換規則適用手続き生成系である．走査手続き生成系の有用性を確認するために，共通アーキテクチャ上のデータ構造変換系を例とし実現する．変換規則適用手続き生成系の有用性を確認するために，入力例と出力例を挙げて説明する．

5.3.1 走査手続き生成系の実現に関する考察

提案した走査手続き生成系の有用性を示すために，共通アーキテクチャ上のデータ構造変換系である EventListener の振る舞いを生成するメタ生成系を実現する．外部イベントの型をツリー構造とした場合，走査手続は標準化したツリーの走査手続を適用する．ツリーの走査手続き生成系は，ツリーの定義を入力とし，ツリーを入力とするデータ構造変換系の走査手続を生成するメタ生成系である．前節で定義したデータ構造のメタモデルから，ツリーを定義し，データ構造の要素毎に変換規則を定義する．

図 6 から定義したツリーと，要素毎の MappingRule を図 8 に示す．

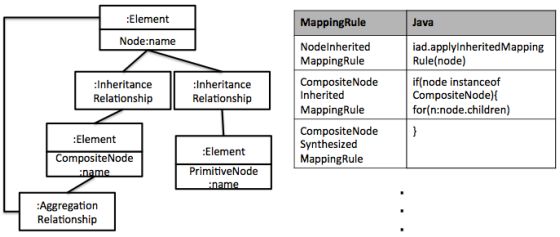

図 8 ツリーの定義と MappingRule

ツリーの定義を入力とする走査手続き生成系の振る舞いを図 9 に示す．

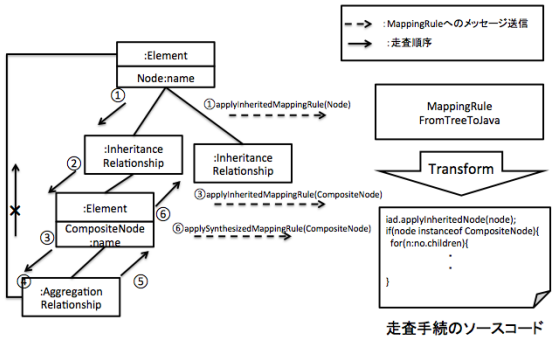

図 9 走査手続き生成系の振る舞い

入力となるツリーの定義は，循環があるので，グラフの走査手続を適用する．子要素を取得して行きがけ順に，要素の名前から適切な相続属性用，帰りがけ順に合成属性用の MappingRule を適用する．

以上の走査手続き生成系の振る舞いによって，データ構造変換系の走査手続きを生成可能であると確認した．

5.3.2 変換規則適用手続き生成系の実現に関する考察

変換規則適用手続き生成系は，意味規則を入力とし，変換規則適用評価を生成し，これもデータ構造変換系と捉える．意味規則はセンテンスとして表現可能なので，走査手続きはセンテンスの走査手続きを適用する．意味規則と出力例を図 10 に示す．

```
.....
if (node instanceof A) {
 AttrSet.add(A.attr);
} else if(node instanceof B) {
 MappingRule mr =
    mrm.getInheritedRule(node.getName());
 mr.setAttribute(B.attr1, A.in);
 queue.enqueue(mr);
}
.....
if(node instanceof B) {
 MappingRule mr =
    mrm. getSythesizedRule(node.getName());
 mr.setAttribute(B.attr2, C.attr);
 queue.enqueue(mr);
}
.....
```

ノード	意味規則
A	A.in = A.attr
B	MappingRule(B.attr1, A.in)
B	MappingRule(B.attr2, C.attr)

図 10 意味規則と出力例

A.in は A ノードの相続属性を表し，A.attr は A ノードが保持する情報を表す．MappingRule は，そのノードの MappingRule を取得し，キューに格納することを表す．MappingRule のパラメータは，その MappingRule を適用する際に必要となる情報を表す．

このように，意味規則に対して評価規則を与えてあげることで変換規則適用評価を生成可能になると考える．

6 おわりに

本研究の背景として，スマートデバイスや Web ブラウザの多様化に伴い，インタラクティブソフトウェアの実行時環境及び開発環境は多様化している．この背景から問題を，インタラクティブシステムを環境に依存する技術を用いて開発することとした．本研究は，共通アーキテクチャのデータ構造変換系を生成する生成系の実現を目的とした．これにより，多様な開発環境と実行時環境の組み合わせに対する開発を容易にする．実現したメタ生成系の妥当性の確認と関連研究との比較を行った．結果として，様々な実行時環境に応じた共通アーキテクチャのコンポーネントが生成可能であり，関連研究に比べ柔軟な実行時環境の追加が可能であることが確認できた．今後の課題としては，現在手書きで書くことを前提としている変換規則の自動生成である．変換規則の HotSpot，FrozenSpot を特定し，アプリケーションフレームワーク化していく必要がある．

参考文献

[1] Allen, S., Graupera, V., and Lundrigan, L. :Pro smart- phone cross-platform development: iPhone, blackberry, windows mobile and android development and distribution. Apress, 2010.

[2] Object Management Group, “OMG Meta Object Facility (MOF) Core Specification,” http://www.omg.org/spec/MOF/2.5/PDF, 2015.

[3] S. J. Mellor, S. Kendall, U. Axel, and W. Dirk, MDA distilled: principles of model-driven architecture. Addison-Wesley Professional, 2004.

[4] 江坂篤侍，野呂昌満，沢田篤史，“インタラクティブソ フトウェアの共通アーキテクチャの提案，”ソフトウェ アエンジニアリングシンポジウム 2015 論文集，2015, pp. 137-144.