Vol.2015-SE-187 No.29
2015/3/13

スマートフォンアプリケーション設計に特化した UML および GUI ビルダによる相互的なモデリング手法

松井浩司[†1]　　松浦佐江子[†2]

モデル駆動開発のベースモデルとして UML モデルが代表的である．UML はシステムの振る舞い・内部構造の設計といった内部設計には向いているが，画面構造およびタッチパネル等によるスマートフォン特有の操作性の設計といった外部設計には向いていない．そこで我々は，各スマートフォン OS の共通機能を纏めた用語集に基づいた UML および GUI ビルダによって作成した「スマートフォン OS に依存しない要求分析モデル」をベースとしたモデル駆動開発を提案してきた．本稿では異なる 2 つのツールで作成する「スマートフォン OS に依存しない要求分析モデル」における各ツール間で共通する設計情報と固有の設計情報について，GUI ビルダによるモデルの変更が与える UML モデルへの影響の観点から事例を用いて報告する．

Mutual modelling method by smartphone application design specific UML and GUIbuilder

KOJI MATSUI[†1]　SAEKO MATSUURA[†2]

UML models are representative models for model-driven development. Though UML is suitable for internal design of a system, it is not suitable for external design such as design of a screen structure and input operation by a touch panel. We have proposed a UML-based model-driven development using a model that is specified by a glossary of smartphone features and a GUI builder that is independent of any specific OSs. In this paper,we discuss how to manage a mutual modeling between UML modeling tool and GUI builder by using a case study.

1. はじめに

モデル駆動開発は，抽象度の異なるモデルを段階的に変換して，ソースコードを生成する技術である．変換の切り口はプラットフォームの考え方に基づき様々である．モデル駆動開発のベースモデルとして UML（Unified Modelling Language）モデルが代表的である．しかし，UML はシステムの振る舞い・内部構造の設計といった内部設計には向いているが，画面構造およびタッチパネル等による操作性の設計といった外部設計には向いていない．理由としては，外部設計に特化したダイアグラムが UML には存在しないからである．UML にはシステムを動的・静的の側面からモデリングをするために様々なダイアグラムが用意されているが，UML は汎用的なモデリング言語であるので，特定のプラットフォームに特化した対象をモデリングする場合，名前・属性・ステレオタイプなどを用いて，その対象の特徴を表現し，モデル化する．つまり，外部設計も UML でモデル化することは可能である．しかし，スマートフォンアプリケーションを対象としたモデル駆動開発[1][2]において, UML モデルを用いた外部設計に関して以下の問題がある．

- オブジェクト図を用いて画面設計を行うため，Widget の位置・大きさ・色や画面構造を視覚的に理解することが難しい．

これは Widget を表したオブジェクトの属性に配置順序などを記述し，オブジェクト間の関連によって画面構造を表現するため，モデルから想定している画面イメージに結びつけることが難しいからである．そのため，想定している画面イメージに近い状態で可視化されたモデルの方が理解し易い．またその他に以下の問題がある．

- スマートフォン特有のタッチパネルによる操作性が外部設計に含まれていない．
- スマートフォンの特長であるアプリケーション連携や多彩なハードウェアの利用も含めた設計をしていないため，スマートフォンの特長を活かしたアプリケーション開発が行えない．
- ユースケースを明らかにし，EntityData のデータ構造および EntityData に対する処理をモデル化していないため，設計した画面構造で実装できる保証がない．

そこで我々は，各スマートフォン OS の共通機能を纏めた用語集に基づいた UML および GUI ビルダによって作成した「スマートフォン OS に依存しない要求分析モデル」をベースとしたモデル駆動開発[3]を提案してきた．我々が提案する手法では Widget の配置指定等の具体的な外部設計は UML を使用せず，タブレット用アプリケーションとして開発する GUI ビルダを用いて視覚的な画面設計およびスマートフォン特有の操作性設計を行う．また各スマートフォン OS の共通機能（Widget，ハードウェアなど）を纏めた用語集を作成し，各モデル要素と対応付けることで

†1 †2 芝浦工業大学
Shibaura Institute of Technology

スマートフォンアプリケーション開発に特化した「スマートフォン OS に依存しない要求分析モデル」の実現を目指した．またユースケース図を作成することでユースケースを明らかにし，各ユースケースに関して EntityData のデータ構造・EntityData に対する処理およびユーザとのインタラクションのモデル化のためにクラス図およびアクティビティ図を作成し， GUI ビルダで扱うモデルにも EntityData の概念を持たせることで，要求分析の段階から実現可能性も検討できるようにした．しかし，各モデリングツールを用いて整合性のあるモデルを作成することは難しい．なぜなら，本稿で提案する GUI ビルダによって作成するモデルは EntityData も扱うため，内部設計を主とした UML モデルとの接点が多くなる．そのため，あるモデリングツールによるモデルの変更は一方のモデリングツールが扱うモデルにも影響を及ぼす可能性が高まる．

そこで本稿では UML と GUI ビルダで作成する「スマートフォン OS に依存しない要求分析モデル」における各ツール間で共通する設計情報と固有の設計情報について，GUI ビルダによるモデルの変更が与える UML モデルへの影響の観点から事例を用いて報告する．

2. 従来手法と本研究のアプローチ

2.1 UML を用いた外部設計アプローチと問題点

スマートフォンアプリケーションを対象としたモデル駆動開発の研究は既に取り組まれているが，外部設計のアプローチ方法は様々である．本稿では従来手法が行っている外部設計のアプローチ方法および問題点に焦点を当てて説明する．

Ayoub ら[1]が提案する手法では，UML モデルであるクラス図とオブジェクト図を用いて外部設計を行う．

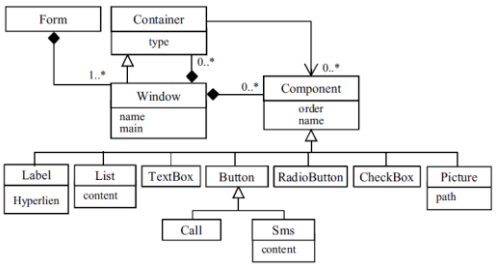

図 1 GUI 部品に関するメタモデル[1]

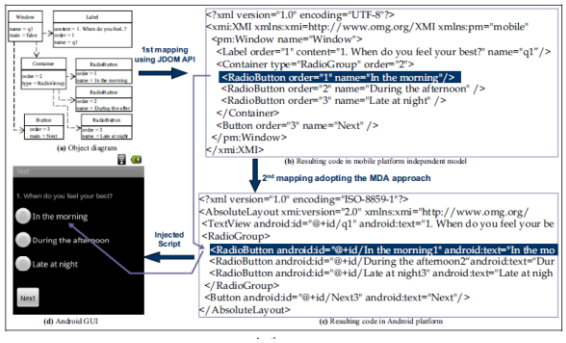

図 2 Android における図 1 の適応例[1]

図 1 および図 2 は[1]から引用してきた図である． 図 1 のメタモデル（クラス図）を元に，画面設計に関わる具体値をオブジェクト図（図 2 の左上）に記述することで画面設計を行っている．具体的には content または name に Widget に表示する文言を定義，order に Widget の配置順序を定義し，オブジェクト図から XMI に変換，その後スマートフォン OS に依存したモデル（図 2 では Android 用 GUI リソースファイル）に変換する．しかし，Ayoub らが提案する手法は，オブジェクト図を用いて Widget の配置順序の指定などを行っているため，モデルから想定している画面イメージを理解しづらい．図 2 のように単純な画面構成でも画面のイメージがし難いため，画面設計においては UML より直感的に設計できる手法が必要である．またタッチパネルによるスマートフォン特有の操作性や外部アプリケーション・センサ等といった連携可能な外部システムを全く考慮していないという問題もある．タッチパネルによる操作や連携可能な様々な外部システムはスマートフォンアプリケーションの活用範囲を大きく広げる重要な特長であるため，開発早期に議論すべき要素である

一方，G.botturi ら[2]が提案する手法はクラス図・オブジェクト図・ステートマシン図を用いて外部設計を行う．

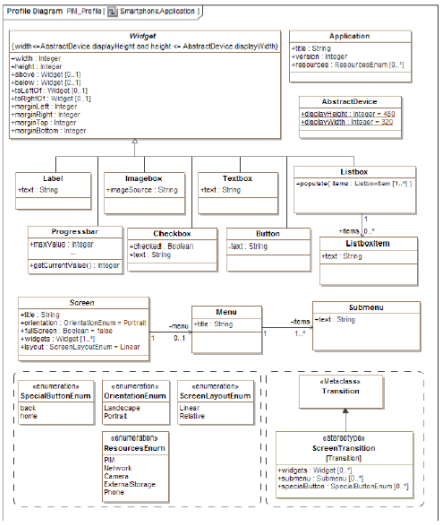

図 3 PIM に関するメタモデル[2]

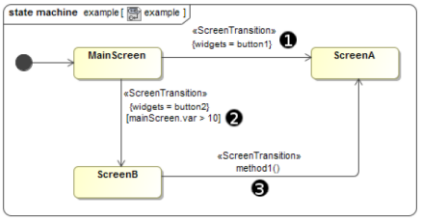

図 4 ステートマシン図による画面遷移図[2]

図 3 および図 4 は[2]から引用した図である．画面設計の方法は基本的に[1]と同様であるが，こちらの手法では図 4 よりステートマシン図を用いて画面遷移順序も定義している．状態が画面に対応し，{widgets=button1}のようにイベントを記述することで画面遷移のきっかけとなるトリガ

ーを定義している．しかし，こちらの手法も[1]と同様にUML を用いて外部設計を行っているため直感的に画面構造を理解できない．また図 3 より Widget 以外のスマートフォンの特長（camera など）がメタモデルに追加されているが，センサや連携できる外部システムなどを使用したアプリケーション開発には対応できない．更に[1][2]と共通する問題として，EntityData のデータ構造および EntityData に対する処理のモデル化について言及されていない．例えば計算結果といった EntityData は Widget を通してユーザに表示するが，ユーザに計算結果を表示したいタイミングに EntityData の存在が保証されていなければ，その外部設計は破綻しているといえる．そのため，外部設計の実現可能性を保証するためには EntityData のデータ構造および EntityData に対する処理のモデル化といった内部設計が必要不可欠であり，逆も然りである．

2.2 本研究のアプローチ

本稿では 2.1 で挙げた問題の解決のために，外部設計には UML を使用せず，直感的に画面設計が行えるツールとして知られる GUI ビルダを独自に開発し，モデリングツールとして外部設計に使用する．2 つ目の問題解決方法としては Widget の他にも各スマートフォン OS で共通するタッチパネルによる操作，連携可能な外部アプリケーション・ハードウェアといった外部システムも分析し，これらを用語集として各モデルのモデル要素と対応付ける．この時タッチパネルによる操作に関しても開発早期に検討できるよう GUI ビルダをタブレット用アプリケーションとして開発することで，画面設計だけではなく操作性の設計に関しても想定した画面イメージで設計が行えるよう実装する．本稿では画面設計および操作性設計を総称して外部設計と呼ぶ．3 つ目の問題解決方法としては EntityData に対する処理を記述するためにアクティビティ図を取り入れ，アクティビティ図およびクラス図を用いて実現可能性を分析する．しかし，特性が全く異なる UML と GUI ビルダの長所を活かしたモデリング方法は難しい．本稿で 2 つのモデリングツールを用いる意図は，外部設計に向いていない UML の代わりに GUI ビルダを用いることである．言い換えれば，内部設計には向いていない GUI ビルダに内部設計を任せることは意図に反するため，各モデリングツールで扱う設計情報は特性に適するように定義しなければならない．そこで本研究では各モデリングツールが共通に扱う設計情報および固有に扱う設計情報を定義することで各モデリングツールの特性を活かすことを目指した．

3. 提案手法

3.1 概要

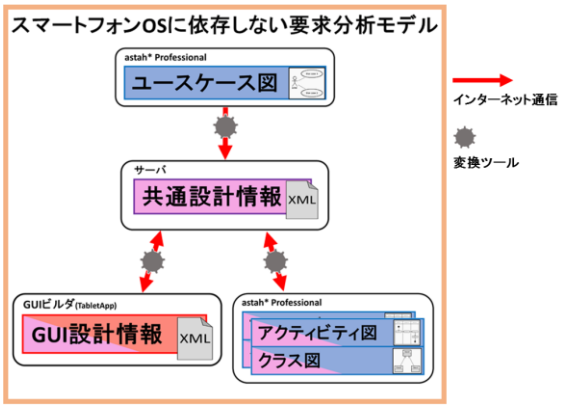

図 5 提案手法の概要

図 5 に提案手法の概要を示す．本稿で使用するモデリングツールは UML モデリングツールである astah* Professional（以下，astah）とタブレット用アプリケーションとして開発する GUI ビルダの 2 つである．そして各スマートフォン OS の共通機能を纏めた用語集と各モデリングツールで作成するモデルのモデル要素を対応付けることで「スマートフォン OS に依存しない要求分析モデル」を実現する．また UML モデルと共通設計情報の変換を自動化するために astah* Plug-in を開発する．「スマートフォン OS に依存しない要求分析モデル」の作成ステップを以下に示す．

1. astah を用いてユースケース図を作成する．
2. GUI ビルダによる外部設計または astah による内部設計を選択する．
3. ステップ 2 で選択したツールでモデリングを行う．
4. ステップ 2，3 を繰り返す．

上記からユースケース図の作成後はツールの使用順序に制約はないことが分かるが，前述で述べた目的から，後述する事例では GUI ビルダ・astah の順でモデリングを行うとする．

3.2 用語集とは

対象としたスマートフォン OS の機能を View，Gesture，State，ExternalSystem とし，各要素が「スマートフォン OS に依存しない要求分析モデル」のモデル要素と対応する．

- View
  画面設計の際，選択できる Widget および Layout である．開発者は 4 種類の Layout と 13 種類の Widget から画面設計を行う．
- Gesture
  InputWidget に対するタッチパネルによるスマートフォン特有の操作を表しており，Single-Touch/Multi-Touch に分類する．

- State
  Foreground はアプリケーションを画面に表示している状態，Background は表示していない状態である．
- ExternalSystem
  連携可能な外部アプリケーション・ハードウェア・データベースを表している．

3.3 共通設計情報とは

内部/外部設計で扱う設計情報は完全に独立していない．例えば UML を用いて内部処理を記述する場合，処理のきっかけとなるトリガーの多くは画面を通したユーザの操作であるため，トリガーとなる Widget およびタップ等の操作方法は明らかにしなければならない．また GUI ビルダによって画面構成を設計する場合，画面を通してユーザに表示する計算結果などの EntityData は何らかの内部処理から得られるものであるため，本当にそのタイミングで期待するデータが生成できるか明らかになっていなければならない．そこで異なるツール間で共通する設計情報を図 6 に示す．

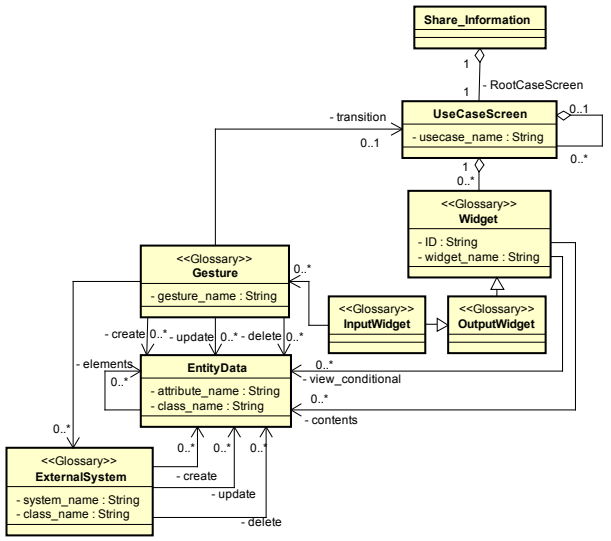

図 6 共通設計情報の定義

一部のクラスに付加されているステレオタイプ（Glossary）は用語集と対応している．内部設計の観点から図 6 を見ると，内部設計で扱う外部設計に関する設計情報は，ID（Widget を識別する文字列）と widget_name（Button など）と Gesture（Tap など）であり，位置・大きさといった視覚的な設計情報は内部設計では扱わない．対して外部設計の観点から図 6 を見ると，CRUD の対象となる EntityData の情報は扱うが，ある EntityData に対してどのような処理をするかといった実現可能性の検討に大きく関わる内部ロジックは扱わない．

以上のように各ツールの長所を活かせるように共通する設計情報を定義し，相互的なモデリング手法を目指す．

4. スマートフォン OS に依存しない要求分析モデルの作成事例

本稿で扱う事例は，本研究室で開発した本学の HP から休講/補講情報を取得し，表示する「休講ナビ」を用いる．また事例で扱う UML モデルはスマートフォンアプリケーション設計に特化するよう拡張しているため，メタモデルと合わせて事例を報告する．

4.1 ユースケース図の作成

ユースケース図にはアプリケーションに要求する機能を定義する．

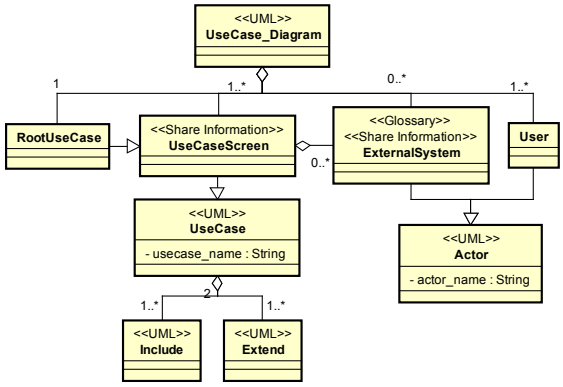

図 7 ユースケース図の定義

まず図 7 のクラスに付加されているステレオタイプの意味について説明する．

- <<UML>>
  一般的な UML のモデル要素であることを示す．
- <<Glossary>>
  用語集と対応していることを示す．つまり ExternalSystem クラスはユースケース図におけるアクターと対応すると解釈する．
- <<Share Information>>
  共通設計情報と対応していることを示す．つまり，このステレオタイプが付加されているクラスの属性（モデル要素）を変更すると一方のモデリングツールで扱うモデル要素も変更される．

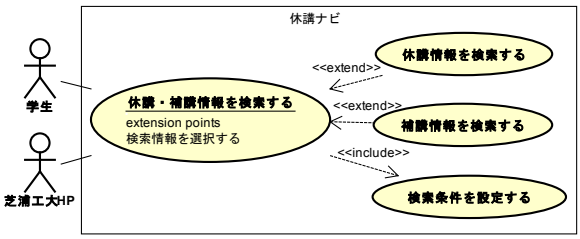

図 8 休講ナビのユースケース図

図 8 は図 7 を元に作成したユースケース図である．図 7 より各ユースケースは UseCaseScreen と定義している．これは各ユースケースが画面と対応していることを意味する．また UseCaseScreen クラスを継承する RootUseCaseScreen

が 1 つだけ存在すると定義しているが，これは開始画面に対応する．RootUseCaseScreen に対応するユースケースの決定ステップを以下に示す．

1. ある 1 つのユースケースに対して，そのユースケースを包含するユースケースまたは，そのユースケースの拡張元となるユースケースを取得する．（複数存在する場合がある）
2. ユースケースを取得した場合，取得したユースケースに対して 1 と同様な操作を行い，収束するまで 1，2 を繰り返し，収束したユースケースを RootUseCaseScreen の候補とする．
3. 1，2 の操作を全てのユースケースに対して行う．
4. 得られた RootUseCaseScreen の候補が全て同一ならば，それを RootUseCaseScreen とする．RootUseCaseScreen の候補が複数存在する場合，関連を見直す．

以上のように開始画面となる RootUseCaseScreen を決定することからユースケース間の関連は画面遷移の順序と対応することになる．

4.2 ユースケース図から生成される共通設計情報

4.1 で作成したユースケース図から生成される共通設計情報を図 9 に示す．

```
<?xml version="1.0" encoding="UTF-8"?>
<UseCase
    usecase_name="休講・補講情報を検索する">
    <InputWidget
        ID=""
        widget_name="">
        <Gesture
            gesture_name="">
            <ExternalSystem>
                system_name="芝工大 HP"
                class_name="HTML “
            </ExternalSystem>
        </Gesture>
    </InputWidget>
    <InputWidget
        ID=""
        widget_name="">
        <Gesture
            gesture_name="">
            <ExternalSystem>
                system_name="芝工大 HP"
                class_name="HTML “
            </ExternalSystem>
        </Gesture>
    </InputWidget>
    <UseCase
        usecase_name="休講情報を検索する">
    </UseCase>
    <UseCase
        usecase_name="補講情報を検索する">
    </UseCase>
    <UseCase
        usecase_name="検索条件を設定する">
    </UseCase>
</UseCase>
```

図 9 ユースケース図から生成される共通設計情報

図 9 の階層構造は図 6 と一致していることが分かる．UseCase に属する属性：usecase_name，ExternalSystem に属する属性：system_name はユースケース図で定義した情報であるから共通設計情報に反映されている．InputWidget に属する属性：ID，widget_name 等はユースケース図では定義していない情報であるため，空である．

4.3 GUI 設計情報の作成

図 9 の共通設計情報を GUI ビルダの入力として起動すると図 10 が表示される．

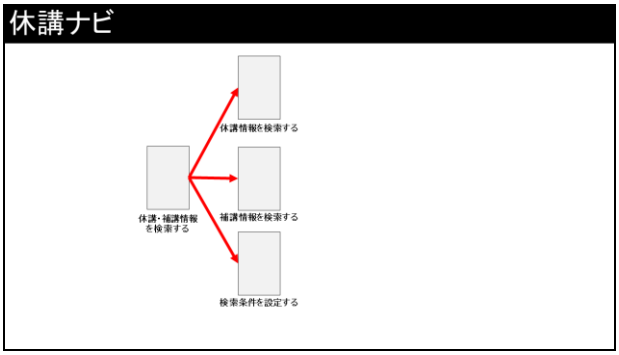

図 10 GUI ビルダ開始画面(イメージ)

ユースケース図で開始画面となる RootUseCaseScreen：休講・補講情報を検索する，および画面遷移の順序を定義したことから図 10 のようにグラフィカルにアプリケーション全体の繋がりを確認できる．しかし，Widget は定義していないため，各画面は空の状態である．

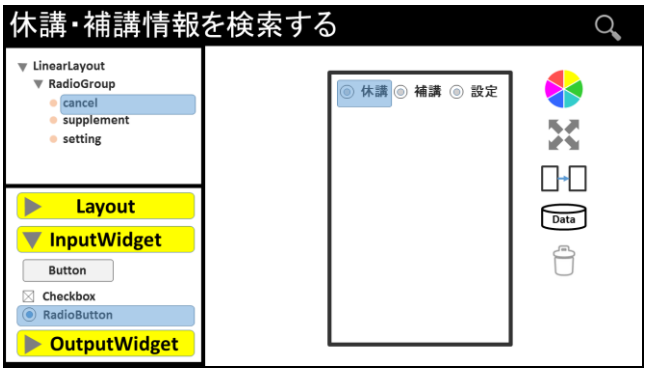

図 11 「休講・補講情報を検索する」の設計画面

図 11 は「休講・補講情報を検索する」の画面設計画面である．左下の View リストから Layout および Input/OutputWidget を選択し，右のデザイン画面にて直感的に画面設計を行う．また画面構造が複雑になり，デザイン画面上で編集したい View が選択し難い場合，左上の View ツリーによって選択操作を容易にする．デザイン画面の右隣にある 4 つのアイコンは上から，色の編集・大きさの編集・画面遷移先の決定・CRUD の定義・View の削除となっている．

ここで 4.4 GUI 設計情報作成による共通設計情報への影響を確認するために以下のことを GUI ビルダで定義することにする．

A) 図 11 のデザイン画面と同様に「休講・補講情報を検索する」に対して RadioGroup を定義し，その中に RadioButton を水平方向に 3 つ定義する．
B) ID は左から cancel・supplement・setting とする．

C)　テキストは左から休講・補講・設定とする.

D)　休講を Tap すると芝工大 HP からクラス名：休講情報を生成し,「休講情報を検索する」に画面遷移する.

E)　補講を Tap すると芝工大 HP からクラス名：補講情報を生成し,「補講情報を検索する」に画面遷移する.

F)　設定を Tap すると「検索条件を設定する」に画面遷移する.

4.4　GUI 設計情報作成による共通設計情報への影響

4.3 で作成した GUI 設計情報による共通設計情報の影響を図 12 に示す.

```
<?xml version="1.0" encoding="UTF-8"?>
<UseCase
    usecase_name="休講・補講情報を検索する">
    <InputWidget
        ID="cancel"
        widget_name="RadioButton">
        <Gesture
            gesture_name="Tap">
            <ExternalSystem
                system_name="芝工大 HP"
                class_name="HTML“>
                <EntityData
                    type=”create”
                    class_name=”休講情報”
                    attribute_name=””>
                </EntityData>
            </ExternalSystem>
            <UseCase
                type=”transition”
                usecase_name="休講情報を検索する">
            </UseCase>
        </Gesture>
    </InputWidget>

    <InputWidget
        ID="supplement"
        widget_name=" RadioButton ">
        <Gesture
            gesture_name="Tap">
            <ExternalSystem
                system_name="芝工大 HP"
                class_name=" HTML “>
                <EntityData
                    type=”create”
                    class_name=”補講情報”
                    attribute_name=””>
                </EntityData>
            </ExternalSystem>
            <UseCase
                type=”transition”
                usecase_name="補講情報を検索する">
            </UseCase>
        </Gesture>
    </InputWidget>

    <InputWidget
        ID="setting"
        widget_name="RadioButton">
        <Gesture
            gesture_name="Tap">
            <UseCase
                type=”transition”
                usecase_name="検索条件を設定する">
            </UseCase>
        </Gesture>
    </InputWidget>

～省略～
```

図 12 GUI 設計情報作成後の共通設計情報

定義 A)～F)による図 9 から図 12 の変更を解説する. まず定義 A)によって RadioButton が 3 つ定義され, 定義 B)によって各 RadioButton に ID が振られた. そして定義 D), E)によって ID=”cancel”および”supplement”の RadioButton と ExternalSystem（芝工大 HP）が関連付けられ, 図 9 の時点では未確定であった 2 つの要素：InputWidget が RadioButton に確定される. また定義 D), E)には RadioButton に対する Gesture と Gesture によって生成する EntityData および遷移する画面先も定義してあるため, これらも同時に更新される. ここで, 定義 A)には”RadioButton を水平方向に”とあるが, 図 6 より共通設計情報は視覚的な情報を一切含まないため, ”水平方向”というデザインに関わる情報は GUI 設計情報が保持し, 共通設計情報には反映されていない. それは定義 C)に対しても同様に言え, テキストは反映されない. また関連付けられなかった残り 1 つの RadioButton は usecase_name=”休講・補講情報を検索する”を値に持つ要素：UseCase 内に新たな要素：InputWidget が更新され, 定義 F)より, この RadioButton に対する Gesture と Gesture によって遷移する画面先が更新される.

4.5　アクティビティ図およびクラス図の作成

アクティビティ図およびクラス図はユースケースと 1 対 1 で対応する. つまり, 1 つのユースケースに対して, 処理の手続きを定義するアクティビティ図およびクラス構造を定義するクラス図を 1 つずつ作成する. 図 13 と図 14 にアクティビティ図の定義, 図 15 にクラス図の定義を示す.

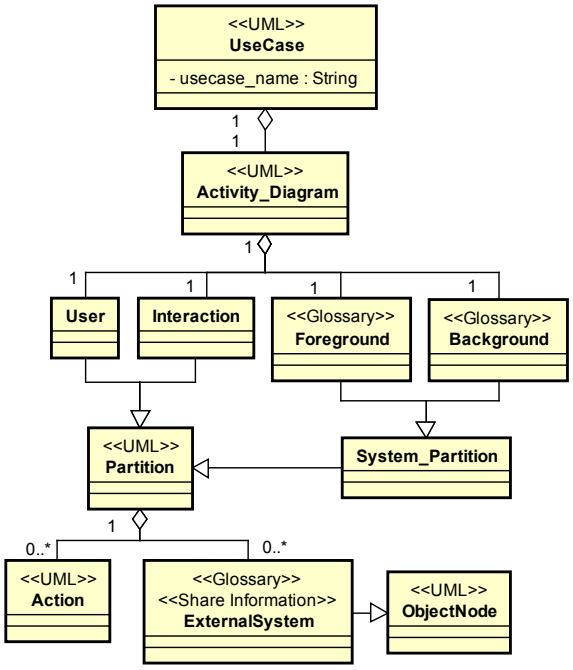

図 13 アクティビティ図を
構成するパーティションの定義

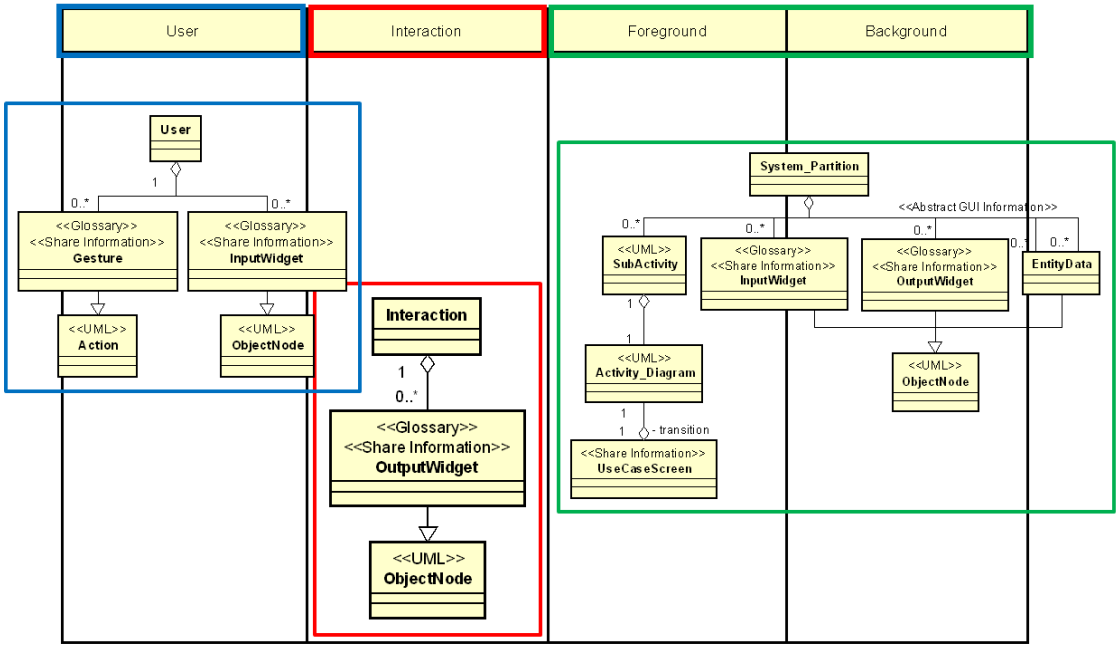

図 14 各パーティションの定義

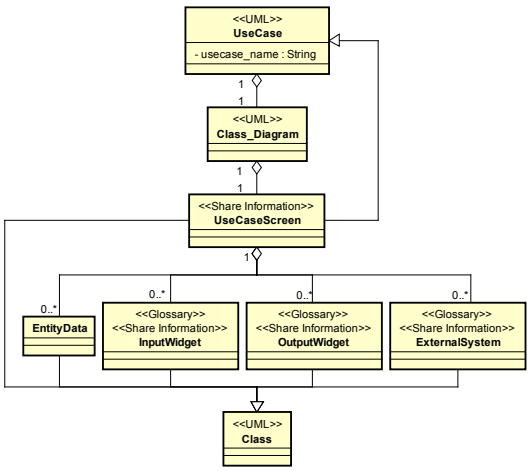

図 15 クラス図の定義

では図 12 から生成されるアクティビティ図とクラス図を図 16 および図 17 に示す．

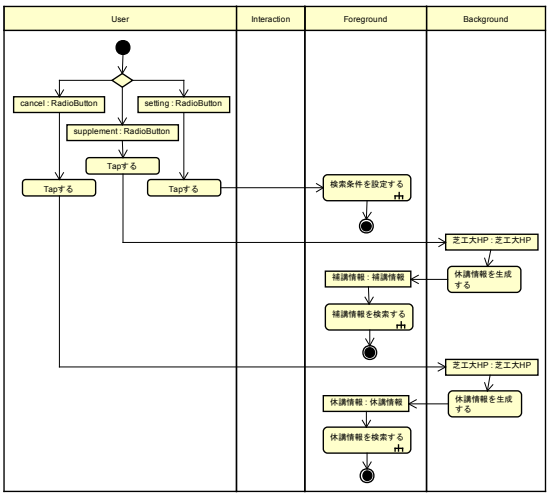

図 16 図 12 から生成されるアクティビティ図

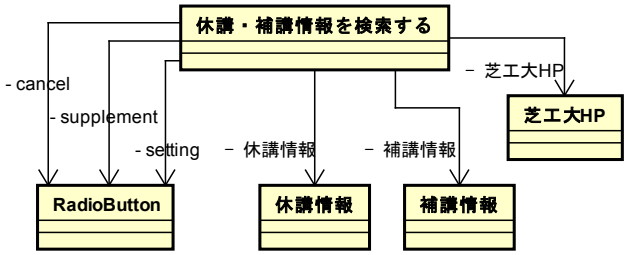

図 17 図 12 から生成されるクラス図

図 16 および図 17 は図 12 から生成される「休講・補講情報を検索する」のアクティビティ図およびクラス図の枠組み（UML スケルトンモデル）である．図 13 より図 16 のパーティションは User・Interaction・Foreground・Background に分割されており，図 14 より User パーティション内のオブジェクトノードは InputWidget に，アクションは Gesture に対応している．ID：cancel，supplement の RadioButton の Tap をトリガーに連携する芝工大 HP は Background パーティション内のオブジェクトノードに，休講/補講情報は Foreground パーティション内のオブジェクトノードに，遷移先の画面はサブアクティビティに対応する．サブアクティビティはアクティビティ図と対応するため，アプリケーション全体の繋がりがアクティビティ図でも定義できる．また図 16 と図 17 のようにアクティビティ図内のオブジェクトノードはクラス図のクラスと対応している．例えば InputWidget クラスは図 14 において ObjectNode クラスを継承しており，図 15 においては Class クラスを継承しているため，このような対応関係になる．ExternalSystem も同様である．

クラス図における共通設計情報との対応関係は図 17 より，widget_name=”RadioButton” がクラス名に，

ID=”cancel”，”supplement”，”setting”がロール名として反映している．また図 6 共通設計情報の定義から明らかであるが，アクティビティ図およびクラス図共に座標・大きさ等の視覚的な情報は反映されていないことが確認できる．

しかし，各 UML スケルトンモデルは要求分析モデルとしては不完全な状態のため，開発者は UML スケルトンモデルに肉付けするように足りない設計情報を補完していく．例えば，それは EntityData に対する具体的な処理である．共通設計情報が保持する EntityData に関する設計情報は，あくまで CRUD レベルの情報であるため，その EntityData を生成するための処理の手続き（公式に基づいた計算手続きなど）といったことは定義されていない．つまり，補完する設計情報とは言い換えれば UML モデルが固有に保持する設計情報である．図 18 に GUI ビルダによる定義と UML モデル要素との対応関係の一部を示す．

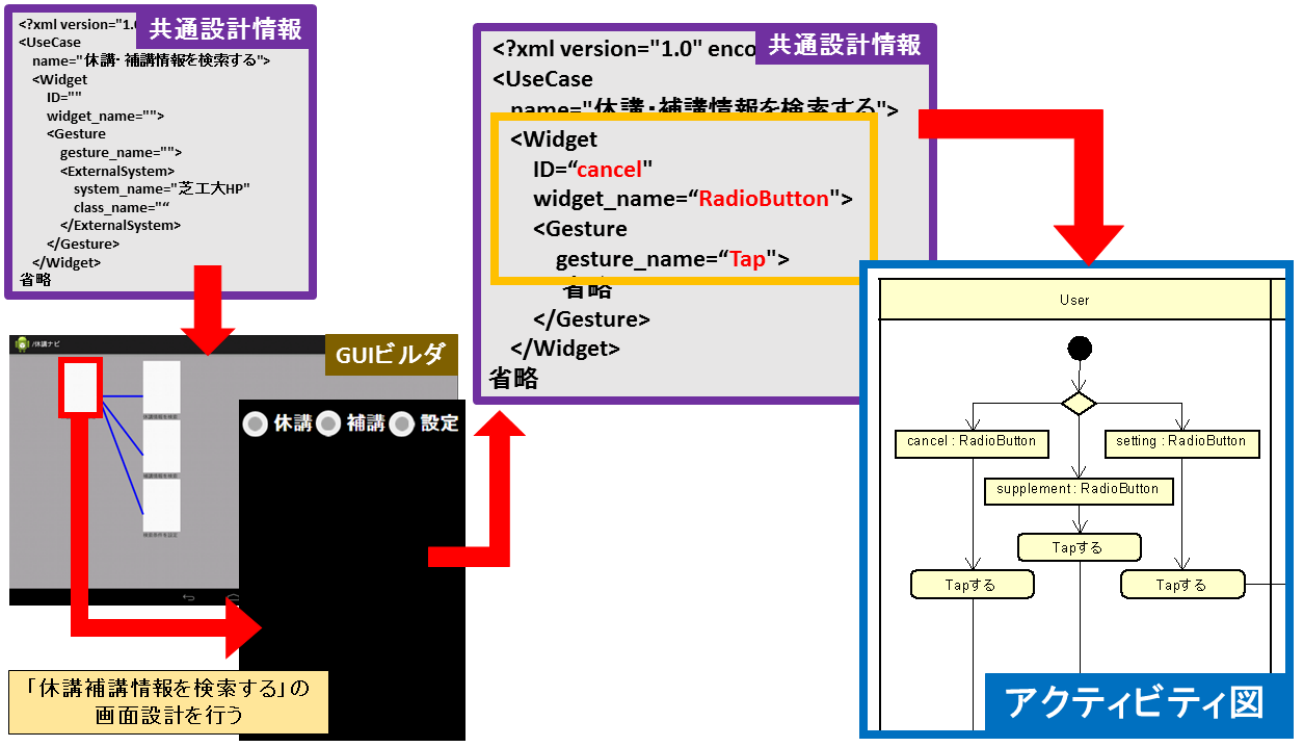

図 18　GUI ビルダによる定義と UML モデル要素との対応関係

5.　おわりに

各スマートフォン OS の共通機能を纏めた用語集に基づいた GUI ビルダと UML で作成する「スマートフォン OS に依存しない要求分析モデル」における各ツール間で共通する設計情報と固有の設計情報について，GUI ビルダによるモデルの変更が与える UML モデルへの影響の観点から事例を用いて報告した．外部設計を UML モデルで行った従来手法[1][2]に対し，本研究ではタブレット用アプリケーションである GUI ビルダを用いることで視覚的な画面設計およびスマートフォン特有の操作性設計の実現，および 4 つの特長に基づいた用語集を作成し，モデル要素と対応付けることで，従来手法よりスマートフォンアプリケーション開発に特化したモデリングの実現を目指した．また EntityData のデータ構造および EntityData に対する処理の分析を目的に加えたクラス図およびアクティビティ図を要求分析モデルに追加することで，開発早期から実現可能性を議論できるようになった．

しかし，GUI ビルダおよび自動化のためのツールは実装していないため，本稿が提案した定義に誤りがある可能性を否めない．そこで今後の課題として，GUI ビルダおよび自動化のための変換ツールを実装し，実際に「スマートフォン OS に依存しない要求分析モデル」を作成することで，定義の見直し改善を行う．

謝辞

本稿に際して，ご協力頂いた皆様に，謹んで感謝の意を表する．

参考文献

1) Ayoub SABRAOUI, Mohammed EL KOUTBI: GUI Code Generation for Android Applications Using a MDA Approach, ICCS, pp.1-6,(2012)
2) G.Botturi, E.Ebeid, F.Fummi, D.Quaglia: Model-driven design for the development of multi-platform smartphone applications, FDL, pp.1-8,(2013)
3) 松井浩司，松浦佐江子: スマートフォン向けアプリケーション設計によるモデル駆動開発手法，情報処理, pp.208-209(2014)