修 士 論 文

組み込みシステムを対象としたソフトウエアアーキテクチャと形式化に関する研究

指導教官　片山 卓也教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

内藤壮司

平成11年 3月 9日

目 次

第 1 章

はじめに

1.1 研究の背景

1.1.1 組み込みシステム

各種の機器に組み込まれてその制御を担う組み込みシステムは、メモリやＣＰＵなどのハードウエアの高機能化によって、適用される応用分野が拡大化している。そのような状況の基で、従来からの組み込みシステム開発法ではその範疇を超えつつある。そこで組み込みシステム開発においても、オブジェクト指向概念を用いた開発アプローチを適用することが注目されつつある。しかし提案されているオブジェクト指向開発法は主に分析工程を中心に展開されており、アプリケーションドメインと依存度の高い設計工程については詳細に言及されていない。特に組み込みシステムの開発では、ハードウエアの構造や実時間性に対する制約が強く、設計の初期の段階で分析工程で抽出された論理的要件を、どのようにして制約を満たすように実装環境に適応していくかが問題となっている。

1.1.2 ソフトウエアアーキテクチャ

ソフトウエアアーキテクチャ[1] には明確な定義が与えられていない。一般にはなんらかの目的意識を持って様々な視点からソフトウエアの構造面をモデル化したものがソフトウエアアーキテクチャであると考えられる。ソフトウエアアーキテクチャをどのような目的意識で捉えるかによって以下に挙げるような種類の構造を見出すことができる。

- 論理構造：ソフトウエアが扱う対象世界や、サービスの構造を表す。どのような概念が用いられどのようなサービスが行なわれるのかは、ソフトウエアの論理的構造から判断することができる。

- 実行構造：ソフトウエア実行時の構造、タスクやプロセスの構造などを表す。ソフトウエアアの構成要素が実行時にどのような振舞いをするのか等はソフトウエアの実行構造から判断できる。

- 開発構造：開発時に認識されるコードやライブラリの構造などを示す。開発時の構成要素やその依存関係、派生関係などは開発構造から判断できる。

またシステム開発においてソフトウエアアーキテクチャは、様々な視点から対象システムの構造を捉え、活用することによりシステム開発工程を少しでも改善させるために用いることができる。ソフトウエアアーキテクチャの活用方法について主な方法を以下に示す。

- 開発手法
  ソフトウエアの開発は、論理構造を踏まえながら適切な実行構造と開発構造を決定する作業であるとみなすことができ、それぞれの記述、決定視点、マッピング等が開発に際しての重要な考慮事項となる。

- 特性評価
  ソフトウエアのいくつかの特性は、ソフトウエアアーキテクチャの側面から評価をすることができる。

- 再利用
  ソフトウエア開発はアーキテクチャ決定の作業と捉える事ができ、アーキテクチャやスタイルの再利用が重要となる。

- 体系化
  ソフトウエアアーキテクチャは、種類、視点、記述、活用法において多様性を持つ。これらの特性を活かすためには、ソフトウエアアーキテクチャを活用を軸にして整理体系化する必要がある。

1.1.3 SES-Based アプローチ

我々は組み込みシステム開発の領域を考察するにあたり、NEC マイコンソフト開発環境研究所の協力を得え、コードレスホン組み込みシステムの開発事例の資料を参照することができた。この開発事例は具体的には、コードレスホンシステムの組み込みソフトウエアに対して、オブジェクト指向開発法 [3][2] に基づいたプロトタイピングを行なったものである [5]。そこで我々は、現場で行なわれた開発手順や問題を、開発担当者とのインタビューや開発ドキュメント (ソースコードを含む) を参照することにより把握した。そしてこれらを基に開発手法を追跡し、系統的に整理し、数学的概念を用いて SES-Based アプローチとして定義した [4]。しかし現状では SES-Based アプローチをシステム開発に適用するまでには至っていない。

1.2 目的

本論文では、組み込みシステム開発におけるソフトウエア構成法の調査として SES-Based アプローチを対象に研究を行なう。そして SES-Based アプローチを用いシステム開発を行ない、評価することで SES-Based アプローチの有効性を明らかにすることを目的とする。

また電話機のシステム開発を SES-Based アプローチの適用対象とすることで、NEC で行なわれたプロトタイピング時の成果物との比較を行なうことができる。よって本論文では SES-Based アプローチに基づいた、コードレスホンシステムの分析/設計モデルの構築を行い、NEC で開発されたシステムとの対応を調べることにより SES-Based アプローチの有効性に関する評価を行ない、SES-Based アプローチ自身にもその結果を反映させることをその目的とする。

1.3 構成

本論文では次のように構成する。第２章では SES-Based アプローチの紹介を行なう。本論文での研究対象である SES-Based アプローチについて述べる。第３章では事例研究としてコードレスホンレスホンシステムの分析を行なう。第４章ではソースコードの解析を行なうことによって、分析で得られた成果物を洗練する。そして SES を抽出し SES-Based 設計モデルを構築する。第５章では考察として、SES、SES-Based 設計モデル、SES-Based ソフトウエアアーキテクチャについて評価を行なう。最後に第６章では本論文のまとめを行ない、今後の課題を示す。

第 2 章

SES-Based アプローチ

本章では、組み込みシステム開発におけるソフトウエア構成法の調査対象である SES-Based アプローチについて述べる。SES-Based アプローチでは、synchronized execution sequence(ses) と呼ばれる同期した処理列を整理するモデルである。このような ses は処理時間情報の単位として扱うことができ、実時間性を考慮した設計の単位とすることができる。このアプローチではインタラクションモデルを SES-Based 設計モデルと呼ばれるモデルに変換する。さらに SES-Based 設計モデルは SES-Based ソフトウエアアーキテクチャと呼ばれるアーキテクチャを持つソフトウエアに反自動変換可能である。以下では SES、SES-Based 設計モデル及び SES-Based ソフトウエアアーキテクチャの定義を行なう。

2.1 SES の定義

まず以下のような基本集合を定義する。

- $ProcessID$:処理識別子の集合
- $EventID$:イベント識別子の集合
- $ObjectID_a$:インタラクションモデルに出現するオブジェクト識別子の集合
- $CondID$:条件識別子の集合
- $StateID$: 状態識別子の集合

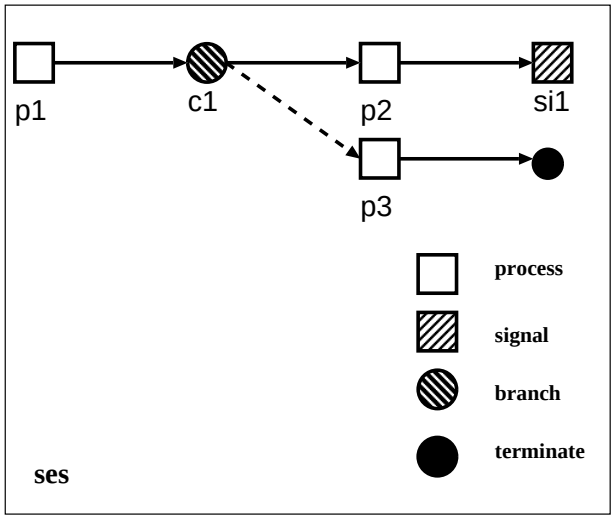
ses は以下のような制約を持つオブジェクト間の通信とそれに伴う一連の処理列と定義する。

- 同一の ses に含まれる処理は逐次に実行される
- ses に含まれる処理の実行は同期して行われる

- ses に含まれる処理は分岐することがあるが、実行されるのは唯一つの命令だけが条件節によって選択され実行される

ses の実行系列 SES_{exp}を以下で定義する。

SES_{exp} = END | NEXT (operation*SES_{exp}) | BRANCH (condition*SES_{exp}*SES_{exp})
$where$ condition $\in$ CondID



DM = (ST_{sys},SES,M_{exp})

$where$ SES $\subseteq$ SESID

また状態遷移図 ST_{sys}は以下に定義される

$ST_{sys} = (S_{sys}, E_{sys}, T_{sys}, s_{sys}, \mathcal{M}_{sys})$

$where$ E_{sys}: SES から送られるメッセージ集合,

$S_{sys} \subseteq StateID$,

T_{sys}: $S_{sys} \times E_{sys} \rightarrow S_{sys}$,

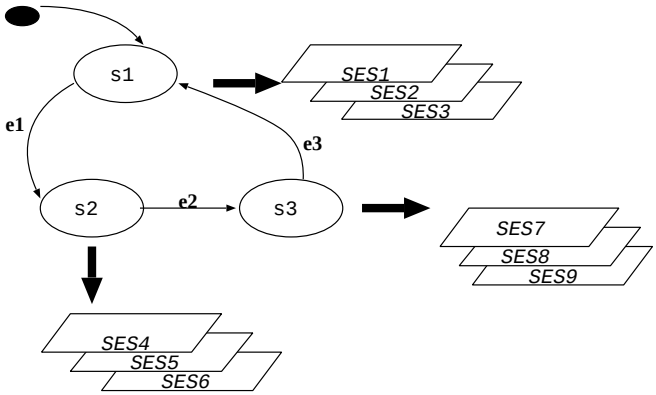
$s_{sys} \in S_{sys}$ (初期状態),

$\mathcal{M}_{sys}$: $S_{sys} \rightarrow$ Pow(SES)

ここで SES の識別子集合 SESID と SES の実行系列 SES_{exp}を対応付ける M_{exp}を導入する。これによって SES の識別子集合は SES が持つ処理列と対応づけられる。

M_{exp}は、SES を識別する識別子、それが表現する処理列と、処理列が発火する条件を対応づけるものである。M_{exp}は以下の定義域と値域を持つ写像である。

$$M_{exp} : SESID \rightarrow SES_{exp} \times CondID$$



2.3 Task

タスクとは現在我々が対象としている、組み込みシステムの実装環境 (RTOS) において、扱われる処理の単位である。RTOS では基本的に複数のタスクを取り扱う。
以下でタスクを定義する。

基本集合に TaskID を追加する
TaskID:タスク識別子集合
各々のタスクは run 状態、ready 状態、wait 状態の 3 つの状態を持っている。
タスクの振舞いを示す状態遷移図 ST_{task}は以下のように定義される。
$ST_{task} = (S_{task}, E_{task}, T_{task}, S_{task})$
where S_{task} = { ready, wait, run }
E_{task} = { sleep, wakeup, disp, undisp }
T_{task}: $S_{task} \times E_{task} \rightarrow S_{task}$
T_{task} = { (wait, wakeup) *mapsto* ready, (run, sleep) *mapsto* wait,
(ready, sleep) *mapsto* wait, (ready, disp) *mapsto* run,
(run , undisp) *mapsto* ready }
S_{task} = wait (S_{task} represents an initial state)

この状態遷移図の定義では、イベント集合 E_{task}を用いており、その要素として、*sleep*, *wakeup*, *disp*, *undisp* を持っている。*sleep* イベントと *wakeup* イベントはタスクが持っているアクションにより発生させられる。

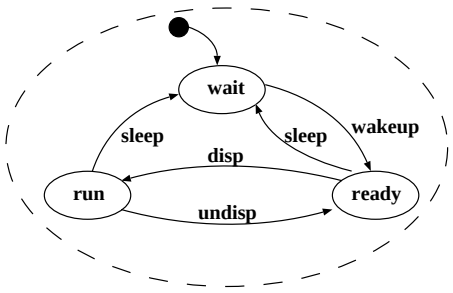


図 2.3: タスクの内部状態

タスクが run 状態で実行する処理について定義する。
タスクはグローバル変数 State を参照することによって、実行する処理を決定する。
タスクが実行する処理は、SES-Based 設計モデルにおける状態 ST_{sys}に付属する ses の集合を用いた状態遷移図、または手続き型の言語のプログラムによって表現される。

タスクとタスクが持つ処理を対応付ける $actionof$ を導入する。$actionof$ は以下の定義域と値域を持つ写像である。

$$actionof : TaskID \rightarrow prog(OP)$$

ここで、$prog(OP)$ は命令集合 OP の要素を用いて構成されるプログラムである。プログラムは C 言語の記述法を用いて定義する。また OP は以下の命令を持つものとする。

- `tslp_tsk`

  引数に時間をとる。自タスクを RUN 状態から watit 状態に移す。引数で指定した時間が経過すると READY 状態になる。

- $objectid.methodid(arg_1, \cdots, arg_n)$ ここで、基本集合として $ObjectID_p$ と $MethodID$ を導入する。$ObjectID_p$ はプログラム中に出現するオブジェクトを識別する、オブジェクト識別子集合である。$MethodID$ はメソッド識別子集合であり、オブジェクトが持つメソッドを識別するものである。この命令は、$objectid \in ObjectID_p$ によって識別されるオブジェクトの持つ、$methodid$ によって識別されるメソッドを引数 $arg_1, \cdots, arg_n$ で呼び出す。

2.4 SES-Based ソフトウェアアーキテクチャ

SES-Based ソフトウェアアーキテクチャにもとづいて実装されるソフトウェア $SOFT$ は、グローバル変数 $state$ の宣言と、タスクの集合 $Task$ により構成される。

$$\begin{aligned} &SOFT = ((Task, actionof), state \text{ の宣言}, \mathcal{IM}) \\ &\text{where } Task \subseteq TaskID, \\ &\mathcal{IM} = \mathcal{IM}_{prc} \oplus \mathcal{IM}_{cond} \end{aligned}$$

2.4.1 $state$ の宣言

SES-Based 設計モデルの状態遷移図による制御を実現する。変数 $state$ は状態遷移図において、システムが現在どのような状態にあるかを表現する。変数 $state$ に関する操作には、以下の命令を用いる。

- $ChangeState$: 引数に状態識別子をとり、グローバル変数 $state$ を引数の状態識別子に更新する。

- $LookState$: 引数はとらずに、グローバル変数 $state$ の値を返す。

これら、2 つの命令は排他制御されているものとする。

2.4.2 タスク

SES に含まれる処理とオブジェクトメソッドの対応を、以下の定義域と値域を持つ写像$\mathcal{IM}_{prc}$により表現する。

$$\mathcal{IM}_{prc} : ObjectID_a \times PrcID \rightarrow ObjectID_p \times MethodID$$

ただし、以下の制約を満たさなければならない。

$$\forall o_1.p_1, o_2.p_2 \in \mathrm{dom}\ \mathcal{IM}_{prc}.o_1 = o_2 \Leftrightarrow$$
$$\mathcal{IM}_{prc}(o_1.p_1) = (o_1'.p_1') \wedge \mathcal{IM}_{prc}(o_2.p_2) = (o_2'.p_2') \wedge o_1' = o_2'$$

又、SES に含まれる条件とその実装の対応を、以下の定義域と値域を持つ写像$\mathcal{IM}_{cond}$により表現する。

$$\mathcal{IM}_{cond} : CondID \rightarrow ObjectID_p \times MethodID$$

各々の SES は Implicit Control Object により実装される。Implicit Control Object は SES に定義されている処理列を実装したものである。

状態 $s(\in StateID)$ に対応づけられている SES $ses(\in \mathcal{M}_{ses}(s))$ により実装される Implicit Control Object を示す。ただし、$\mathcal{M}_{exp}(ses) = (sesexp, cond), \mathcal{IM}_{cond}(cond) =$`o.cond` であるものとする。

```
if (o.cond){
   SES_PROG;
   tslp_tsk(tmout);
}else{
   tslp_tsk(tmout);
}
```

`SES_PROG` は $sesexp$ に以下の構成規則を適用することにより構成される。`tmout` には、短い時間を用いる。

- $NEXT(oid.prc, exp)$ の時

  `oid.prc() ;`
  `exp` に構成規則を適用したもの;

  ただし、$\mathcal{IM}_{prc} =$`(oid.prc)` であるものとする。

- $NEXT(eid, exp)$ の時、

```
ChangeState (s');
```
exp に構成規則を適用したもの

ただし、$s' = T_{sys}(eid, s)$ であるものとする。

- $BRANCH(c, exp1, exp2)$ の時

```
if (oid.c){
   exp1 に構成規則を適用したもの
}else{
   exp2 に構成規則を適用したもの
}
```

ただし、$\mathcal{IM}_{cond}(c)$ =oid.c であるものとする。

- ENDの時、規則の適用を終了する。

状態 $s(\in StateID)$ に対応づけられている SES $ses(\in \mathcal{M}_{ses}(s))$ を上の手法を用いて実装した Implicit Control Object のプログラムを $ICO(s, ses)$ と書くことにする。

それぞれのタスクは、複数の SES を実現する。タスク $t \in TaskID$と、それにより実現される SES 集合の対応を、写像 IM_{task}により表現する。$\mathcal{IM}_{task}$を以下の定義域と値域を持つ。

$$\mathcal{IM}_{task} : Task \rightarrow Pow(SES)$$

ただし、$\mathcal{IM}_{task}$は以下の 2 つの制約を満たさなければならない。

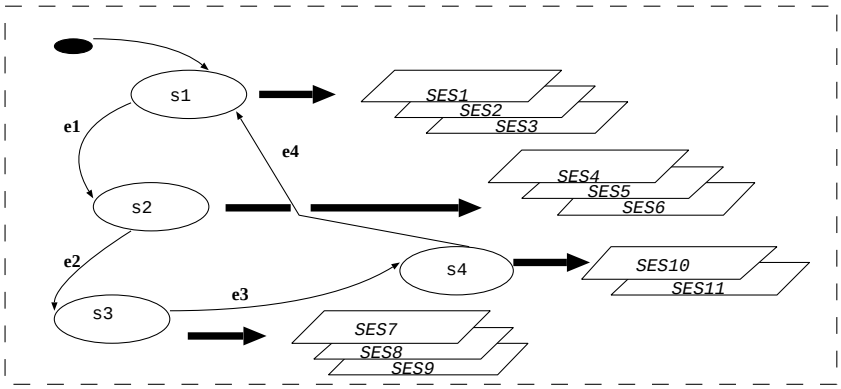
$$\forall ses \in SES.\exists t \in Task.ses \in \mathcal{IM}_{task}(t)$$

$$\forall s \in S_{sys}.\forall ses_1, ses_2 \in \mathcal{M}_{sys}(s).\forall t \in Task.\neg(ses_1 \in \mathcal{IM}_{task}(t) \wedge ses_2 \in \mathcal{IM}_{task}(t))$$

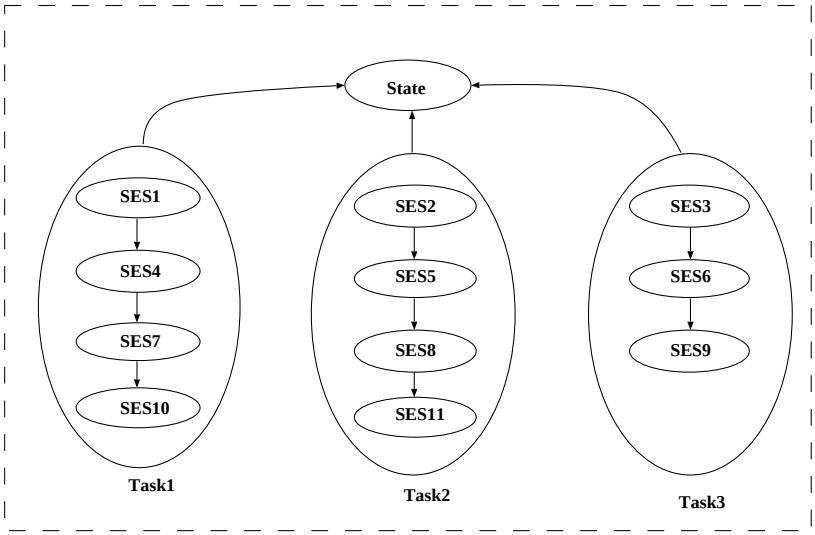
$\mathcal{IM}_{task}(t) = ses1, ses2, ...$であるとき、タスク t のプログラム $actionof(t)$ は以下のアーキテクチャを持つものとして実装される。ただし、$ses1 \in \mathcal{M}_{sys}(s1)$, $ses2 \in \mathcal{M}_{sys}(s2)$ であるものとする。

```
while (1){
   switch (LookState()){
     case s1:
        ICO(ses1, s1) で構成されるプログラム
        break;
```

```
    case s2:
        ICO(ses2, s2)で構成されるプログラム
        break;
.......
    }
}
```



SES Based 設計モデル



タスクの構造と**SES**の対応

第 3 章

コード レスホンシステム開発事例

SES-Based アプローチでは、システム分析の成果物として得られるインタラクション図から、処理列の基本単位となる SES を抽出する。しかしインタラクション図の作成に至る開発工程については、言及されていない。しかしこのアプローチの基礎になっている事例研究では OOSE のアプローチに沿った分析／設計工程を適用している。そこで本研究ではコード レスホンシステムの、システム分析に OOSE を採用する。OOSE を採用することによって、プロトタイピングの際に使われたドキュメントや成果物をより効果的に活用することができる。

OOSE におけるシステム開発の手順は、分析工程において要求仕様よりユースケースと呼ばれる処理の単位を抽出する。次にユースケースをシステムが行なう処理の基本単位として、オブジェクトの抽出を行ないインタラクション図を設計する。

3.1 事例研究の方針

本研究において事例対象とするコード レスホンシステムについて説明する。コード レスホンシステム開発のための要求仕様として、実際のコード レスホンが製品化された際に用いられることになるユーザーズマニュアルを用いる。その他にはシステムに組み込まれるハード ウエアの仕様や、電話回線網を利用するための仕様等が存在する。

ハード ウエアの仕様は実装時のインタラクションのレベルまであらかじめ決定されており、その多くはシステム開発の初期の段階で電話機に組み込まれることが決定される。また電話回線網を利用するためのインタラクションの方法等に関する仕様も同様に決定されている。
これらの仕様を利用するには、その領域に関する専門的な知識を要し我々が今回行う研究の範疇を超えている。

しかし我々が参照することができた資料の中には、オブジェクト指向方法論を適用し設計されたソースコード (機能の一部を実現している) が含まれていた。そこで本研究では以下の二つを手

法を用いて、システム開発の事例研究を行なう。

- ユーザーズマニュアルを仕様としたシステム分析／設計を行なう。
- 実装されたソースコードの構造を分析することにより、システムの分析を行なう。

これは実装されたコードの構造を分析することで、システム分析／設計で明らかにできなかった、コードレスホンのシステムの特性を調べることを目的にしている。

3.2 シナリオの作成

OOSE に基づき、コードレスホンの親機通常着信処理について、シナリオを構築する。以下にコードレスホンの通常着信処理に関する要求仕様をユーザズマニュアルより抽出する。またここで行なわれた抽出結果の番号付けは、マニュアルの段落構成を基にして作成した。

3.2.1 着信処理

1. - 外線通話以外の状態。(子機が外線通話中は、親機のスピーカー／保留ランプが点滅)
2. - 回線が外部の接続先から電話機に接続される。
   - 外部スピーカーから着信音発生。
     同時にスピーカー／保留ランプ (赤色) 点滅、ダイヤルランプ (緑色) が順番に点滅する。
3. - 受話器を上げる。
   - 着信音が止まる。
   - 受話器スピーカー、マイクを介して接続先と通話を行う。
     スピーカー保留ランプが点灯。
4. - 回線との接続を断つ場合は受話器を降ろす。
     同時にスピーカー。保留ランプが消灯 (待機状態)

3.2.2 発信処理

1. - 端末利用者が受話器を上げると、スピーカ／保留ランプ (赤色)、ダイヤルランプ (緑色) が点灯する。
2. - 電話回線との接続が正常なら、受話器スピーカから発信音が流れる。
   - 端末利用者は発信音を聞くと、数字キーを用いて接続先の電話番号を入力する。
   - 数字キー、スイッチなどはボタンを押すたびに音を鳴らす。
3. - 入力した電話番号が無効の場合、そのことを示すメッセージが流れる。
4. - 交換機によって電話回線が接続先に接続されると、受話器のスピーカーから、呼出音が聞こえる
5. - 接続先が応答すると通話状態になる。

- スピーカー、マイクを用いて接続先と通話を行う。

6. - 交換機によって電話回線が接続先に接続されると、受話器のスピーカーから、話中音が聞こえた場合、接続先が他の機器 (電話回線) と接続中であることを示す。
   - 利用者が電話機を置くと回線が断たれる。

7. - 通話終了、または接続先が通話中の場合。受話器を降ろすことで回線が断たれる。

   - スピーカ／保留ランプが消灯する。

3.3 オブジェクトの抽出

要求仕様であるシナリオから抽出されるオブジェクトの候補を次に挙げる。

- 電話機本体、ランプ (バックライト)、キー、ベル、スピーカー、マイク、受話器、回線、電話機利用者、回線利用者

次に問題ドメインオブジェクトモデルを作成する。

- 電話機の利用者との入力インターフェイス記述から、抽出される抽象的なキーオブジェクトとキーを継承するフック、数字キー:図 3.1

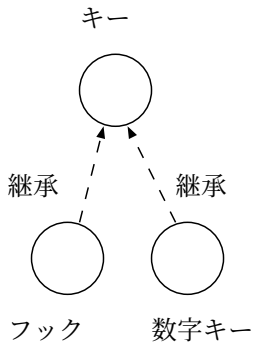


図 3.1: ドメインオブジェクト：キー

- 電話機の利用者との通話インターフェイス記述から、抽出される抽象的な受話器オブジェクトと受話器を継承して通話に関するそれぞれの機能に特化したマイク、スピーカー:図 3.2
- 電話機の利用者との情報出力部記述から抽出される抽象的な出力部オブジェクトと出力部を継承してそれぞれの機能に特化したベル、バックライト：図 3.3

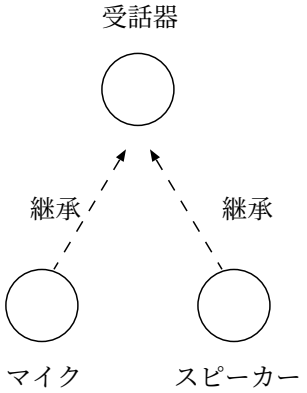


図 3.2: ドメインオブジェクト：受話器

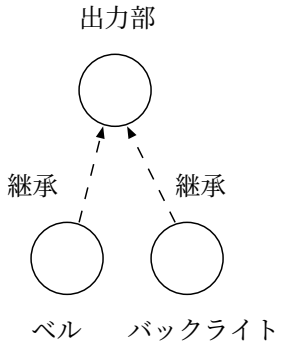


図 3.3: ドメインオブジェクト：出力部

ただしこの段階でのモデルはあくまで、最初の分析から作られたもので詳細な機能の実現など今後修正される可能性は残っている。次にアクターを識別する。

- 電話機利用者、回線

3.4 ユースケースの抽出

前節で識別したアクターについて考察し、電話機に対してどのような操作を行うのか調査する。電話機利用者をアクターとしてユースケースを識別する。また図 3.4に電話利用者と、識別された着信処理ユースケースの振舞いを示す。

- 通常着信処理
  - 利用者が利用者 IF のフックをオフフックにする。
  - 通話する。
  - 利用者が利用者 IF のフックをオンフックする。(通話終了)

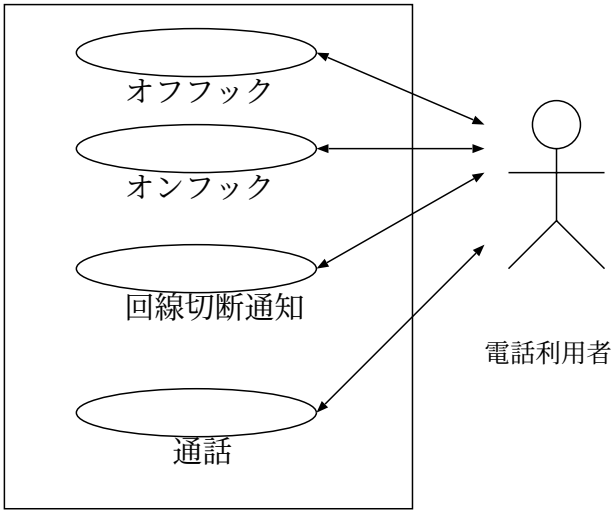


図 3.4: 利用者から識別した着信処理ユースケース

発信処理に関して電話利用者をアクターとしてユースケースを実現する。図 3.5に電話利用者と、識別された発信処理ユースケースの振舞いを示す。

- 通常発信処理
  - 利用者が受話器をあげる。
  - 通話先番号を入力。
  - 通話する。
  - 利用者がフックをオンフックする。(通話終了)

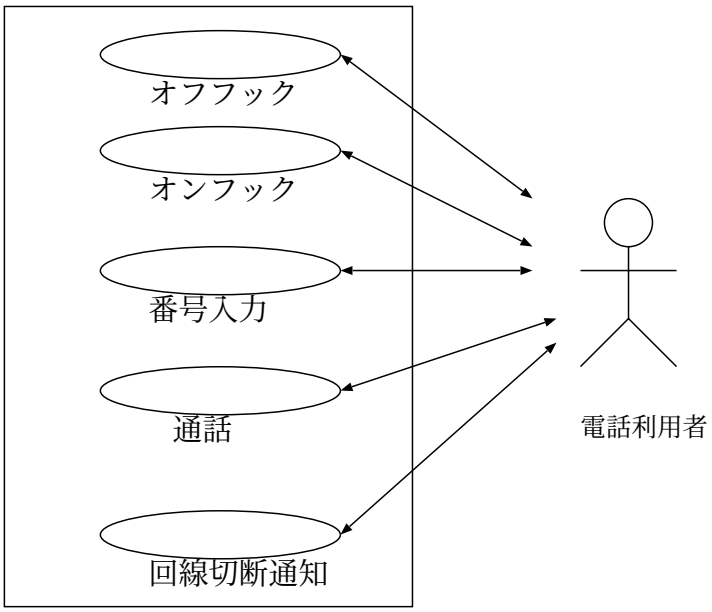


図 3.5: 利用者から識別した発信処理ユースケース

次に回線をアクターとしてユースケースを識別する。また図 3.6に回線と、識別した着信処理に関するユースケースの関係を示す。

- 通常着信処理
  - システムに外線が着呼を知らせる。
  - 通話する。
  - 利用者による回線切断を通話先利用者に知らせる。(通話終了)
  - 通話先利用者が、回線を切断したことをシステムに知らせる。(通話終了)

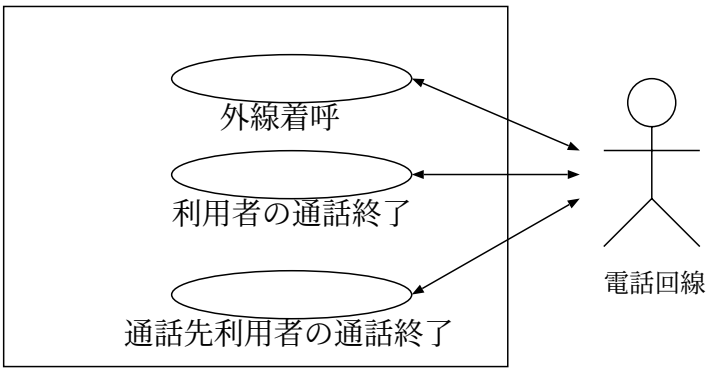


図 3.6: 回線から識別した着信処理ユースケース

回線をアクターとみなして発信処理ユースケースを識別する。図 3.7に回線と、識別した発信処理に関するユースケースの振舞いを示す。

- 通常発信処理
  - 利用者の接続要求中に、接続先が応答したことを知らせる。
  - 通話する。
  - 利用者による回線切断を通話先利用者に知らせる。(通話終了)
  - 通話先利用者が、回線を切断したことをシステムに知らせる。(通話終了)

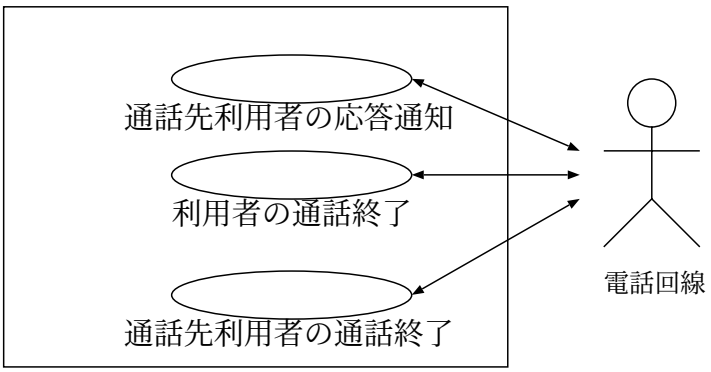


図 3.7: 回線から識別した発信処理ユースケース

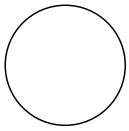
3.5 分析オブジェクトの作成

ユースケースで表現された振る舞いを、ドメインモデルのオブジェクトに割り当て、どのユースケースにどのオブジェクトが責任を持つのかを明確にする。まずユースケース着信処理、ユースケース発信処理に関する、領域オブジェクトを識別する。

- 親機 (電話機)、ランプ (バックライト)、キー、ベル、スピーカー、マイク、受話器、回線、電話機利用者、回線利用者

3.5.1 実体オブジェクト

次に図 3.8要求仕様から導かれる、実体オブジェクトを示す。



電話機

図 3.8: 実体オブジェクト

3.5.2 分析オブジェクトとユースケースの関係

ユースケースに記述した振る舞いとオブジェクトの関係を明確にする。

外線着呼ユースケース

- 外線が着呼する。
  - 着呼を受け取る回線インターフェイスオブジェクト (回線 IF) が必要である。
- システムは着呼をしらせるベルを鳴らす。バックライトが点滅。
  - 回線インターフェイスオブジェクトの状態を取得して、ベルを鳴らすインターフェイスオブジェクト、バックライトを点滅させるインターフェイスオブジェクトが必要である。しかし、回線インターフェイスオブジェクトの状態を解析するのはインターフェイスオブジェクトは適切ではないと考え、制御オブジェクト回線ハンドラを導入する。
- 利用者が電話機のフックをオフフックにする。

  - 電話機インターフェイスオブジェクト (フック) が必要である。

- バックライトが点灯、通話状態にする。

  - 電話機インターフェイスオブジェクトの状態を取得して、バックライトを点灯させるインターフェイスオブジェクト、通話に必要な受話器インターフェイスオブジェクト、マイク IF、スピーカー IF を動作させる。しかしインターフェイスオブジェクトが他のインターフェイスオブジェクトの状態を解析するのは適切ではないと考え、制御オブジェクト、電話利用者ハンドラを導入する。

- 利用者が電話機のフックをオンフックする。

  - 電話機インターフェイスオブジェクト。

- オンフックするとバックライト消灯、通話終了。

  - 電話機インターフェイスオブジェクトの状態取得は、すでに導入された、電話利用者ハンドラによって行なわれ、バックライトインターフェイスオブジェクトに消灯、受話器インターフェイスオブジェクト、マイク IF、スピーカー IF に動作終了させる。

- 回線先利用者が回線切断。(通話終了)

  - すでに導入された、回線ハンドラによって回線インターフェイスオブジェクトの状態を取得。バックライトインターフェイスオブジェクトに消灯、電話利用者ハンドラに回線切断を知らせる。

通常発信ユースケース

通常発信処理ユースケースに関して、その振舞いとオブジェクトの関係を明確にする。

- アクターが受話器をあげる。(オフフック)

  - 電話機インターフェイスオブジェクト (フック IF) が必要である。

- バックライト点灯。マイク、スピーカー回線接続。

  - フック IF の状態を取得して、バックライトを点灯させるインターフェイスオブジェクト (バックライト IF)、受話器インターフェイスオブジェクト (マイク IF、スピーカー IF) が必要である。しかしフック IF の状態を解析するのは、インターフェイスオブジェクトは適切でないと考え、制御オブジェクト電話利用者ハンドラを導入する。

- アクターがキーを押す。

  - 電話機インターフェイスオブジェクト (キー IF) が必要である。

- 発信先が応答をすれば、発信通話開始。
  - 通話先利用者の応答を受け取る回線インターフェイスオブジェクト (回線 IF) が必要である。

- オフフック後アクターが受話器を置く。(オンフック)
  - フック IF が必要。

- オンフックすると、ライト消灯、通話終了。
  - フック IF の状態取得はすでに導入された、電話利用者ハンドラを用いる。バックライト消灯、通話終了にバックライト IF、マイク IF、スピーカー IF を用いる。

- 回線先利用者が回線切断。(通話終了)
  - すでに導入された、回線ハンドラによって回線インターフェイスオブジェクトの状態を取得。バックライトインターフェイスオブジェクトに消灯、電話利用者ハンドラに回線切断を知らせる。

ここまでの分析作業で、ユースケース毎の分析オブジェクトを得た。これら全ての分析オブジェクトをまとめて、分析モデルを得る。
図 3.9に分析モデルを示す。
この段階で要求分析で得た、問題ドメインオブジェクトモデルを参照し、要求分析の結果と大きく異なる点がないか確認する。

図 3.9において制御オブジェクトの動作に着目すると、制御オブジェクト間で二つのインターフェイスオブジェクトを制御し、制御オブジェクト間では直接メッセージ通信を行なっているのがわかる。そこで制御オブジェクトの動作を一つにまとめる役割を持つオブジェクトを導入する。それによりオブジェクト間の関係が、理解しやすいものとなる。図 3.10にあらたに制御オブジェクト、ユースケースハンドラを導入した分析モデルを示す。

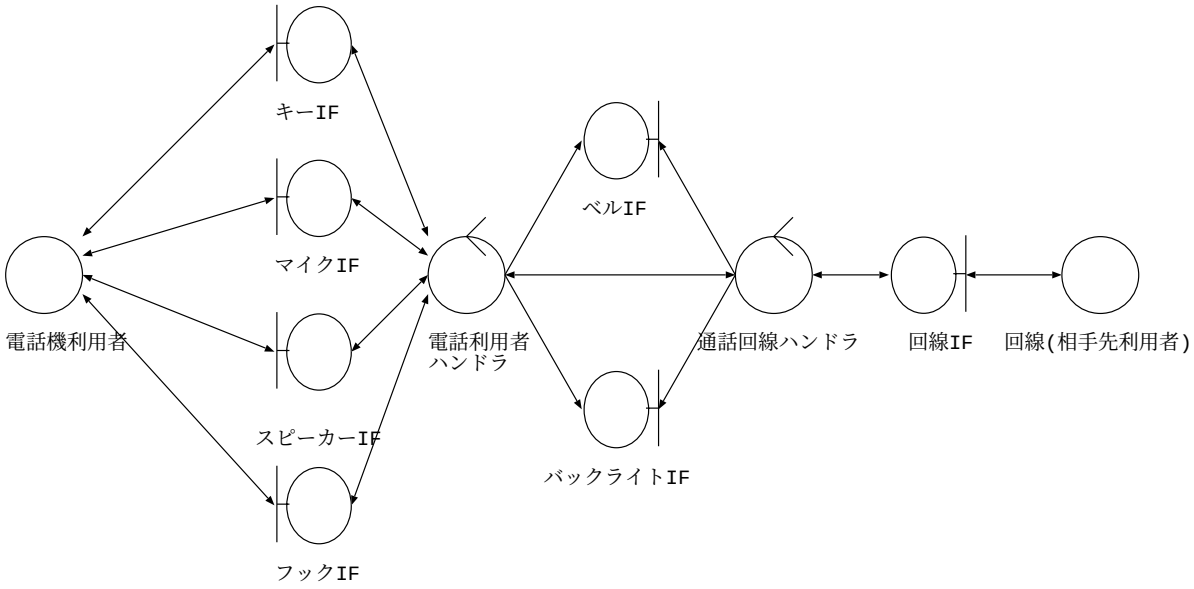


図 3.9: 発信／着信処理ユースケースの分析モデル

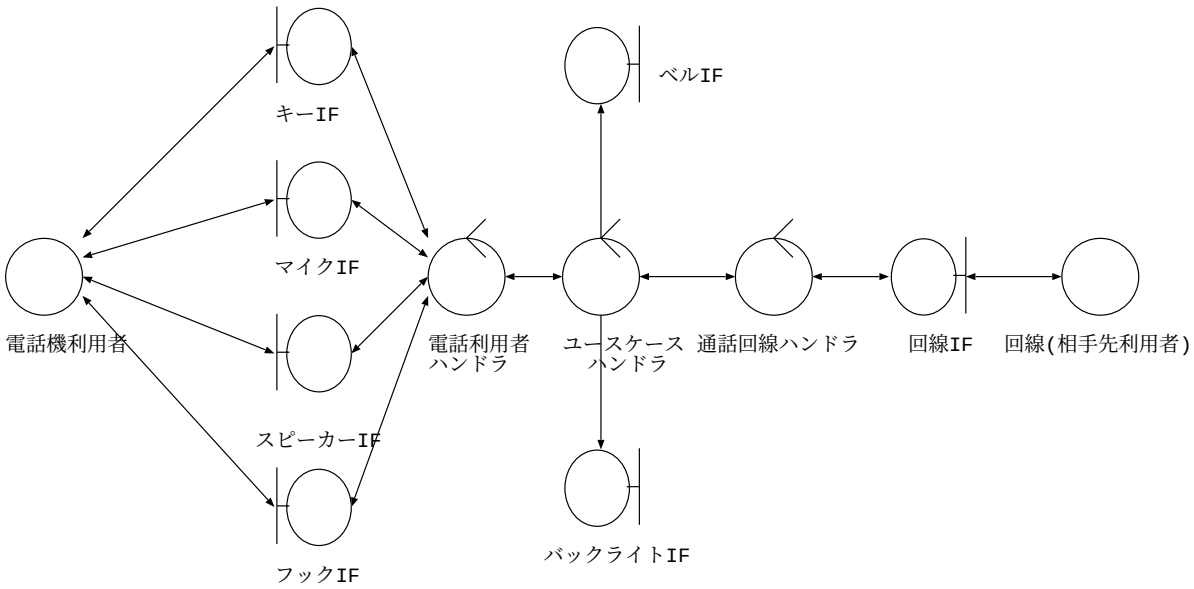


図 3.10: 修正された分析モデル

3.6 インタラクション図の作成

本節では電話機の組み込みシステムの分析工程で得られた成果物などからインタラクション図の設計を行なう。またこれまでの分析工程のアプローチは OOSE に基づいた分析方法を適用してきた。

しかし設計工程におけるインタラクション図の記述法については、OOSE に定義されている記述法を拡張したものを適用する。これは後の工程で、インタラクション図から SES を抽出すること、SES から SES-Based 設計モデルへの変換の際にインタラクション図を利用しやすいようにすることを目的とするためである。よって本節ではまずインタラクション図の記法について拡張を行なう。そして拡張した記法に従って、分析結果からインタラクション図を作成する。

3.6.1 インタラクション図

前節までで作成した分析結果を用いてインタラクション図を作成する。インタラクション図の表現形式を用いることは、システムの構造を理解する上で、ユースケースを中心とした解析が行ないやすいというメリットがある。
またシステムの機能面の構造をインタラクション図で表すことは、SES-Based 設計モデルとソースコードとの対応を調べるときにも有用である。

次にインタラクション図を作成するにあたり、本研究で用いるインタラクション図について新たに定義を行なう。なぜならインタラクション図の表現形式には様々な細かな違いがあるため、コードの構造をインタラクション図で表すためには、詳細な定義を必要になるためである。本研究で利用するインタラクション図は OOSE のユースケースで用いられるものである。しかし OOSE においても表現の自由度を保つため詳細な定義は行なわれていない。そこで本研究では OOSE のインタラクション図を基に、より詳細な定義を行なう。以下ではまず OOSE におけるインタラクション図の定義を示す。

OOSE においてインタラクション図はブロックがお互いに刺激を送りあうことによるコミニュケーションを記述する。このインタラクション図の基盤はユースケースである。それぞれのユースケースにおいてどのように刺激が送られ、どのような順序で送られるかを詳細に記述する。このようにブロック間で送られる刺激の列によってユースケースが記述される。

ここでブロックは書かれるコードの抽象としてみなされなければならない。またそれによりブロックとソースコードの間には追跡可能性が高くなることが望ましい。

またブロックがクラスであって、幾つかのインスタンスがある場合は、個々インスタンスを個別に並べて書いてもよい。最も見やすいインスタンスを１つ書くのもよい。インタラクション図では記述したい物の外界を表す縦線を書く。これはシステム境界と呼ばれる。

インタラクション図では時間は縦に下の方向に進む。またインタラクション図の時間軸は時間の長さに比例したものではなく、制御された事象と見なければならない。また事象間の距離は事象間の実際の時間とは無関係である。

インタラクション図の左端、システム境界の左側には、そのシーケンスを記述する。この記述は文章であり、構造化された文あるいは疑似コードで構成される。
またこの文章はインタラクション図で記述したユースケースので何が起こるのかを記述する。この部分は操作と呼ばれる。操作はインタラクション図において四角形で表し、ブロックの縦線に書き込む。文章による記述は (後で実装される) 操作が行なわれるブロックに属する。

インタラクション図の文法とセマンティクスは実際の実装言語に従わなければならない。

3.6.2 インタラクション図の記法

前節では OOSE で定義されるインタラクション図を整理したが、ここでインタラクション図の記法を定める。これはインタラクション図から SES-Based 設計モデルへの変換をより行ないやすくするためである。OOSE におけるインタラクション図の主な特徴を整理し以下に示す。インタラクション図において、縦の線はオブジェクトを表し、横の線は刺激を表す。そこで図中の刺激及びオペレーションの識別を行なうために、図中に識別子を導入する。

- p：インタラクション図においてオブジェクトで実行される処理の識別子とする。
- e：インタラクション図において刺激が送るイベントを表す。

インタラクション図における刺激には様々な意味が考えられる。たとえばプロセス間通信であるとかプロセス内通信であるなどである。プロセス内の刺激であれば、通常一つのプロセス内での呼出である。よって刺激についての適切な定義は実装環境に依存する。本研究においては、刺激の明確な定義は行なわない。ただし作成したインタラクション図についての説明を行う場合、分析段階で判明していることがあればその都度イベントの説明箇所で記述する。また縦軸のブロックはオブジェクトの処理が行なわれている事を示し、ブロックは操作が呼ばれるときに、生成され処理が終わるとブロックも終わる。

OOSE におけるインタラクション図では、システム境界の外側にはインタラクションのシーケンスを記述することができる。この記述については、インタラクション図のブロックで実行されるユースケース名を記述する。これらの定義によって、インタラクション図を作成する際、オブジェクト間のインタラクションからは、SES を抽出することができ SES-Based 設計モデルへの変換をより行ない易くすることができる。

図 3.11に本研究で用いるインタラクション図の記法例を示す。図 3.11では A,B がオブジェクト A,B を表し、p1,p3 はオブジェクト A で実行されるオペレーションを表し p2 はオブジェクト B で

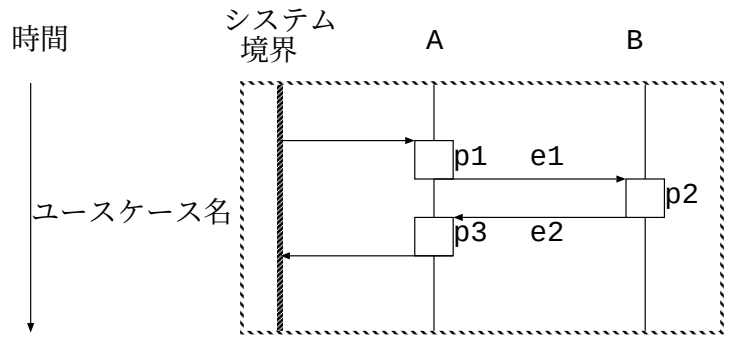


図 3.11: インタラクション図の図的表現

実行されるオペレーションを表す。このインタラクションは SES の定義と一対一に対応する。具体的にはオブジェクト間のメッセージ通信が、メタなレベルではどのように振舞われるのかを表現できるように拡張を行なう。これによって SES-Based 設計モデルにおける状態遷移図の作成が見通しの良いものとなり得る。インタラクション図よりかなり具象的な (実装に近い) ものが得られることは明らかである。
よってコードの構造をできるだけ抽象度の高い視点からみなせるように構造を示すことに重点を置く。

3.6.3 着信処理ユースケースの修正

前節までで行なった定義に従ってまず、これまでに得た分析結果からインタラクション図を作成する。すると分析結果からインタラクション図に変換できないユースケースが発見された。変換できなかった理由はフック IF と回線 IF に関するユースケースでフックの ON/OFF と、回線の接続／切断を組み合わせたユースケースが存在したことを分析段階では発見できなかった為である。さらなる分析の結果、フックの ON/OFF は回線の開放/閉結を意味することが判明した。回線の閉結とは、外部からの着信要求を受けないようにすることであり、回線の開放とは外部からの着信要求を受け付ける状態であることを示す。

また回線の接続要求とは、外部からの接続要求を意味する。これはシステムがオンフック時のときは、外線着呼になりオフフック時のときの接続要求とは、番号発信先の応答を意味する。

そこでユースケースの記述を以下のように修正する。電話機利用者をアクターとしてユースケースを識別する。また図 3.12に電話利用者と、修正後の着信処理ユースケースの関係を示す。

- 通常着信処理

– 利用者が利用者 IF のフックをオフフックにする。
– 回線閉結。
– 回線接続要求開始。
– 利用者が利用者 IF のフックをオンフックする。
– 回線要求終了。
– 回線開放。

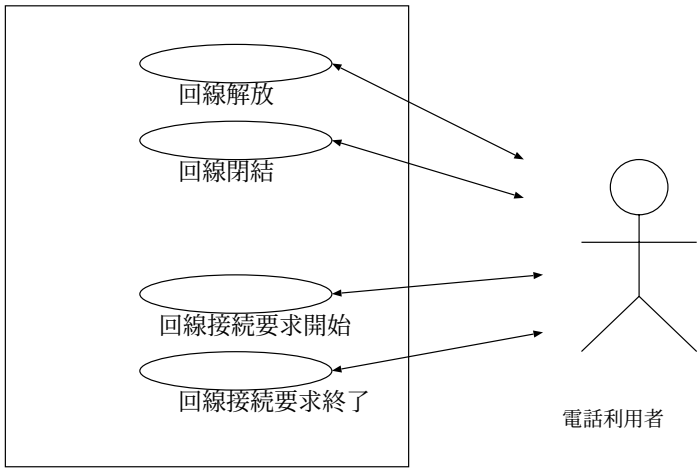


図 3.12: 利用者から識別した着信処理ユースケース

また変更に基づき発信処理ユースケースを修正し以下に示す。図 3.13に電話利用者と、修正後の発信処理ユースケースの関係を示す。

- 通常発信処理
  - 利用者がフックをオフフックする。
  - 回線閉結。
  - 通話先番号を入力。
  - 回線接続開始要求。
  - 利用者がフックをオンフックする。
  - 回線接続終了要求。
  - 回線開放。

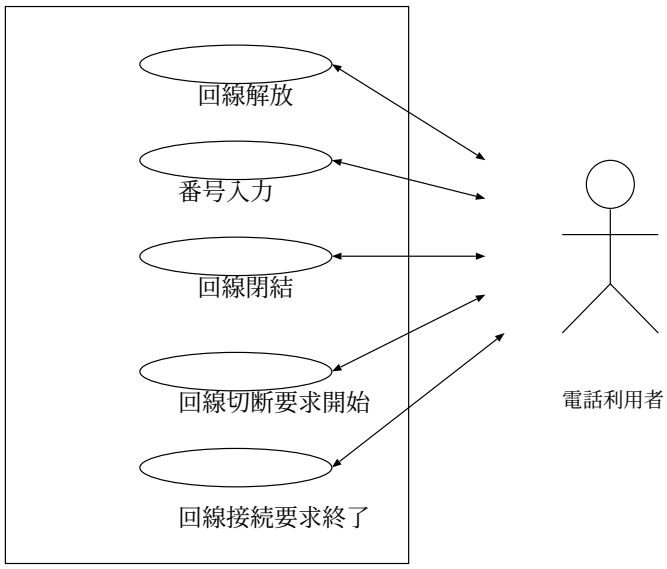


図 3.13: 利用者から識別した発信処理ユースケース

次に回線をアクターとしてユースケースを識別する。図 3.14に回線と、識別した着信処理に関する修正後のユースケースの関係を示す。

- 通常着信処理
  - 回線接続開始要求。
  - 回線接続要求終了。
  - 回線閉結。
  - 回線開放。

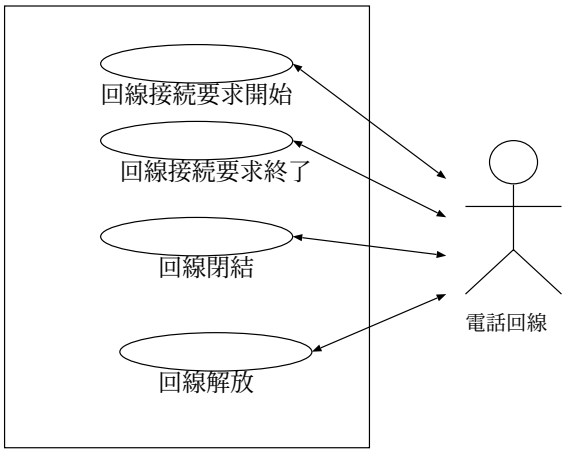


図 3.14: 回線から識別した着信処理ユースケース

図 3.15に回線と、識別した発信処理に関する修正後のユースケースの関係を示す。

- 通常発信処理
  - 回線開放。
  - 番号発信。
  - 回線接続要求開始。
  - 回線接続要求終了。
  - 回線閉結。

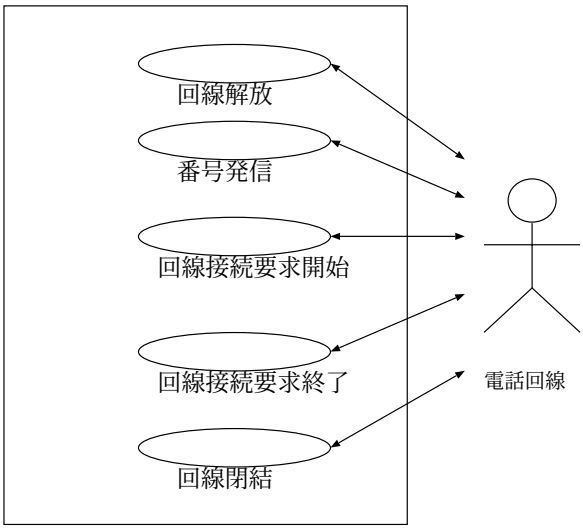


図 3.15: 回線から識別した発信処理ユースケース

3.6.4 ユースケースのインタラクション図作成

着信処理ユースケース

ここで修正されたユースケースと、分析段階で得られた成果物を基にして、インタラクション図を作成する。図 3.16は着信処理に関するユースケースにおいて特に電話利用者ハンドラ、通話

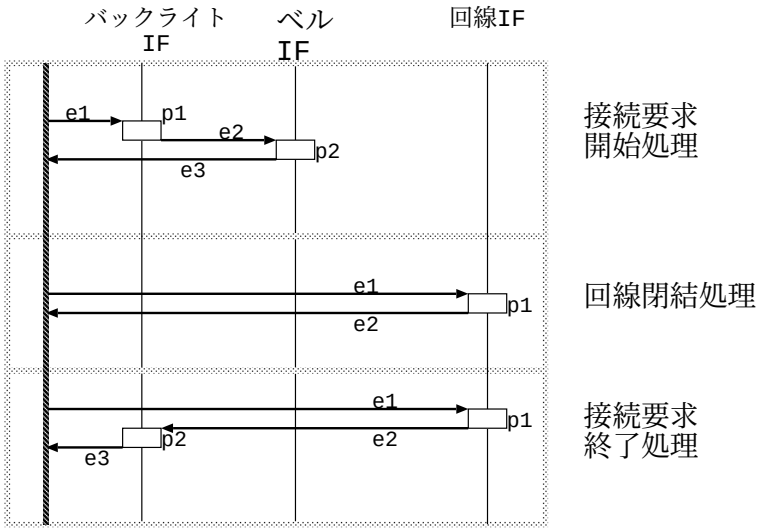


図 3.16: 着信処理に関するユースケース

回線ハンドラ、回線 IF のオブジェクト間のメッセージ通信に重点をおいてインタラクション図を作成したものである。
この図において斜線で囲まれた枠は修正されたユースケースを表す。以下で図 3.16のインタラクションについて説明する。

- 接続要求開始処理
  - p1：バックライト ON
    バックライト IF において実行されるバックライトを点灯する処理を表す。
  - e1：バックライト ON 要求
  - e2：バックライト ON 要求終了通知
    これはユースケースハンドラから送られる e1 というイベントによって起動され e2 と

いうイベントを送出することによって処理の終了をベルに伝える。ベルはバックライト ON が終了した通知を受け取りベルを ON にする。

- p2：ベル ON
  ベル IF において実行されるベルを鳴らす処理を表す。
- e3：ベル ON 要求終了通知
  ユースケースハンドラに終了を伝える。

- 回線閉結処理
  - p1：回線閉結処理
    回線 IF において実行される回線閉結処理を表す。
  - e1：回線閉結要求
  - e2：回線閉結要求終了通知
    回線ハンドラから送られる e1 というイベントによって処理が起動され、e2 というイベントを送出することで処理の終了を回線ハンドラに伝える。

- 接続要求終了処理
  - p1：回線開放処理
    回線 IF において実行される回線開放処理を表す。
  - e1：回線開放処理要求
  - e2：回線開放処理要求終了通知
    これは回線開放処理が、回線ハンドラから送られる e1 というイベントによって起動し、処理の終了を e2 のイベントを送ることによって伝える。バックライト IF は回線開放処理が終了を伝えたことによってバックライトを OFF にする。
  - p2：バックライト OFF
    バックライト IF において、バックライトを消すという処理を表す。
  - e3：バックライト OFF 処理終了通知
    e3 というイベントをユースケースハンドラに送ることによって終了を伝えている。

図 3.17は着信処理ユースケースにおいて、特に電話利用者ハンドラ、マイク IF、スピーカー IF に注目してインタラクション図を設計したものである。インタラクション図はシステム分析の結果を用いて作成されている。図 3.17の場合は着信処理ユースケースにおいてオフフック処理、オンフック処理についてオブジェクト間のインタラクション及び、実行される操作を表している。

- オフフック処理

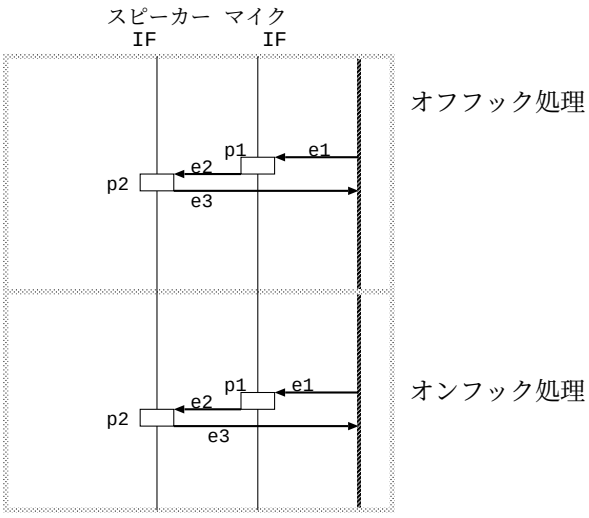


図 3.17: 着信処理に関するユースケース

- p1：マイク ON
  マイク IF によって実行されるマイク ON という処理を表す。
  e1：マイク ON 要求 e2：マイク ON 要求終了通知
  これは電話利用者ハンドラによって送られる e1 というイベントによって起動され、処理の終了を e2 というイベントをスピーカー IF に送ることでスピーカーを ON にする。
- p2：スピーカー ON
  スピーカー IF によって実行されるスピーカー ON という処理を表す。e3：スピーカー ON 要求終了通知
  e3 というイベントを送ることで電話利用者ハンドラに処理の終了を伝えている。

- オンフック処理

  - p1：マイク OFF
    マイク IF によって実行されるマイク OFF という処理を表す。
    e1：マイク OFF 要求 e2：マイク OFF 要求終了通知
    電話利用者ハンドラによって送信されるイベント e1 によって起動され、処理終了を e2 というイベントをスピーカー IF に送ることによって伝え、スピーカー IF は e2 のイベントを受け取りスピーカー OFF を実行させる。

– p2：スピーカー OFF

スピーカー IF によって実行されるスピーカー OFF という処理を表す。

e3：スピーカー OFF 処理要求終了通知

e3 というイベントを電話利用者ハンドラに送ることによって処理の終了を知らせる。

発信処理ユースケース

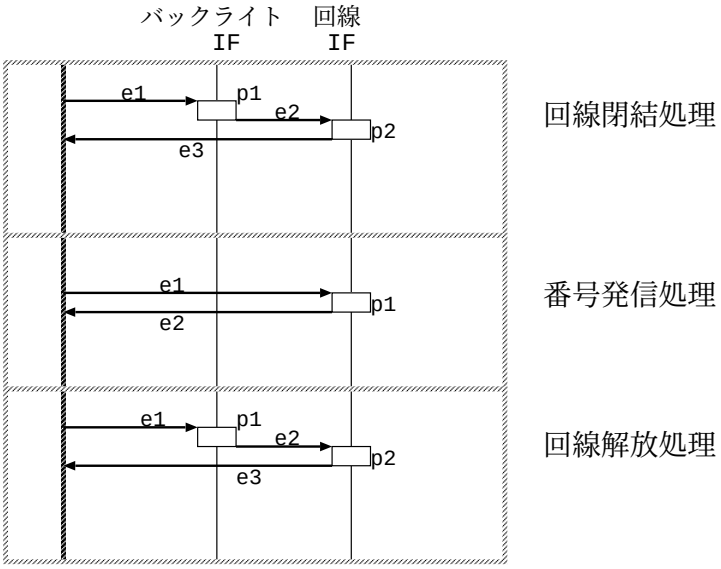


図 3.18: 発信処理に関するユースケース

図 3.18は発信処理ユースケースに相当する部分を、制御オブジェクト（利用者ハンドラ、回線ハンドラ）に注目して作成した図である。ここでも着信処理ユースケースと同様、分析段階で得た成果物を用いて修正されたユースケースを表現している。

- 回線閉結処理
  - p1：バックライト ON
    バックライト IF において実行される、ライトの ON という処理を表す。
  - e1：バックライト ON 要求
  - e2：バックライト ON 要求終了通知
    これはユースケースハンドラから送られる e1 というイベントによって起動し、処理の終了を e2 というイベントを回線 IF に送ることで知らせる。回線 IF は e2 というイベントを受け取り、回線閉結処理をさせる。
  - p2：回線閉結処理
    回線 IF において実行される、回線閉結処理を表す。
  - e3：回線閉結要求終了通知
    処理の終了を e4 というイベントを回線ハンドラに送ることで伝える。
- 番号発信処理

- p1：番号発信処理
  回線 IF において実行される番号発信という処理を表す。
- e1：番号発信処理要求
- e2：番号発信処理要求終了
  e1 というイベントを回線ハンドラから送られることで起動する。そして e2 というイベントを回線ハンドラに送ることで処理の終了を知らせる。

- 回線開放処理
  - p1：バックライト OFF
    バックライト IF において実行されるバックライト OFF という処理を表しこれはユースケースハンドラによって送信されるイベント e1 によって起動される。イベント e2 によって終了を伝える。
  - e1：バックライト OFF 要求
  - e2：バックライト OFF 要求終了通知
    p2 は回線 IF において実行される回線開放処理を表す。これは e1 というイベントを回線ハンドラから送られることで起動し、e2 というイベントを回線ハンドラに送ることで処理の終了を通知する。

図 3.19は発信処理ユースケースの処理の中で、主にインターフェイス部（フックＩＦ、マイクＩＦ、スピーカーＩＦ、キーＩＦ）に着目して作成したインタラクション図である。

- オフフック処理
  - p1：マイク ON
    マイク IF によって実行されるマイク ON という処理を表す。
  - e1：マイク ON 要求
  - e2：マイク ON 要求終了通知
    これは電話利用者ハンドラによって送られる e1 というイベントによって起動され、e2 というイベントをスピーカー IF に送ることで終了を通知している。スピーカー IF は e2 というイベントを受け取りスピーカーを ON にする。
  - p2：スピーカー ON
    スピーカー IF によって実行されるスピーカー ON という処理を表す。
  - e3：スピーカー ON 要求終了通知
    e3 というイベントを送ることで電話利用者ハンドラに処理の終了を伝えている。

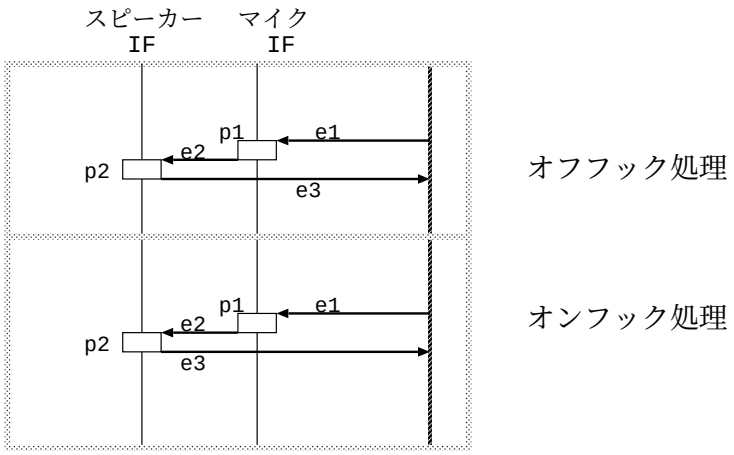


図 3.19: 発信処理に間するユースケース

- オンフック処理
  - p1：マイク OFF
    マイク IF によって実行される処理マイク OFF を表す。
  - e1：マイク OFF 要求
  - e2：マイク OFF 要求終了通知
    電話利用者ハンドラによって送られるイベント e1 によって起動される処理で、e2 というイベントを電話利用者ハンドラに送ることで処理の終了を知らせる。

第 4 章

ソースコードの解析

前章では分析結果をもとにインタラクション図を作成した。そこで実際にシステムの一部の機能の振舞いを実現したソースコードの構造を分析する。そして分析結果を用いて拡張したインタラクション図を用いてユースケースを表現する。そして前章までの分析/設計結果と比較することによって、これまでの設計工程では明らかにできなかったシステムの特性を抽出することを試みる。

4.1 インタラクション図の修正

ソースコードを分析し、オブジェクトとオブジェクト間の通信がどのような手順で行なわれているのかを中心に調べ、その結果を基にして分析モデルを作成する。

図 4.1はコードの構造から抽出した分析モデルである、分析段階で作成した図 3.10とは異なる構造が得られた。その主な違いを以下で示す。

- あらたに導入された制御オブジェクト
  - アナログ回線ハンドラ
    フックの状態によってマイク IF、スピーカー IF と外部回線との接続に関する制御を行なう。
- オブジェクト間の通信関係が異なっていたところ
  - アナログ回線の接続を制御するアナログ回線ハンドラが導入されていた。
  - バックライト IF は電話利用者ハンドラ、ユースケースハンドラ、通話回線ハンドラから制御される。

これらの違いは、分析段階での考慮が十分でない、設計段階で出た問題点がオブジェクト間の通信スタイルに影響を与えることなどが原因となって起こり得る。この例ではアナログ回線を制

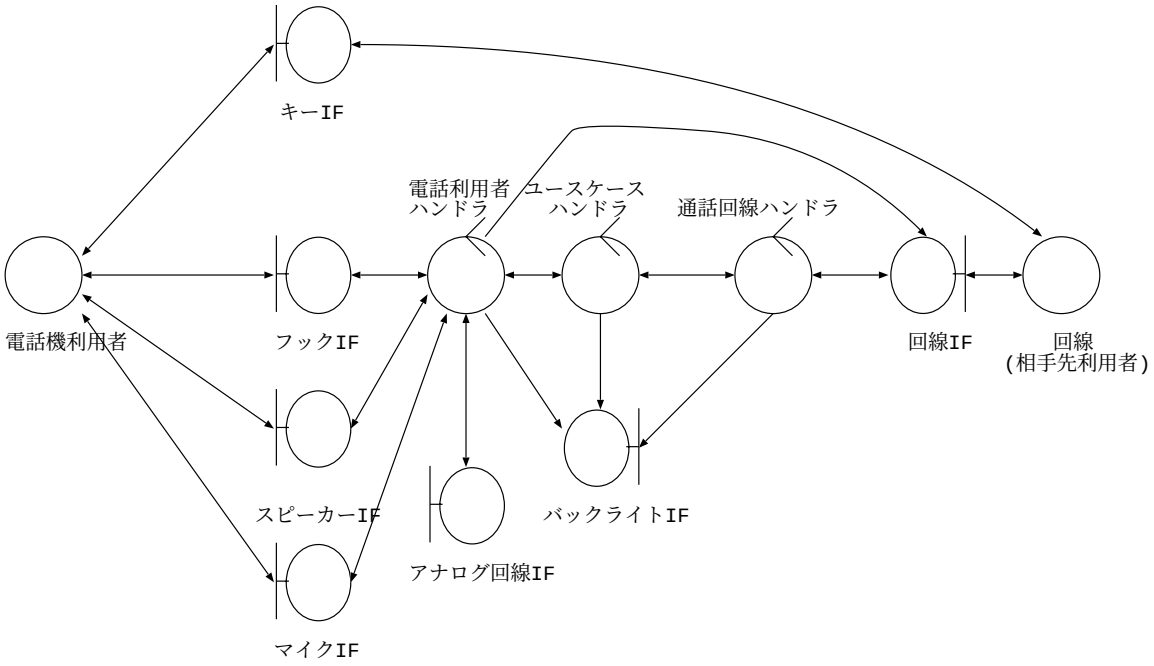


図 4.1: 分析モデル 2

御するアナログ回線ハンドラが導入されている。これはマイクやスピーカー、電話回線などの回線同士を接続あるいは切断する処理を行なう。これは電話機のハードウエアの性質について、知識がある設計者でないと分析段階では発見することは困難となる問題であると予想される。

バックライト IF が 3 つの制御オブジェクトと通信する理由について考える。制御オブジェクトがバックライト IF と関連を持つ理由を示す。通話回線ハンドラにおいては、接続要求を知らせるためと、着呼中の場合点滅させるという機能を実現させるために通話回線ハンドラがライトの ON ／ OFF を繰り返し行なっていた。しかしこれは実装上の都合と判断できる。

電話利用者ハンドラでは、フックの動作に同期してバックライトの制御を行なうという機能を実現していた。ユースケースハンドラは他の 2 つの制御オブジェクトとその動作が重複していたため、特別の理由を見出すことができなかった。しかし実際にはバックライトを制御する機能は、コードの洗練の結果一つのオブジェクトの機能としてまとめられていたことで理解した。

番号発信については、最初に行なった分析では、利用者ハンドラがキー IF の制御を行なっていたが、実際にはキー IF から送られてくる刺激は、ハードウエアの処理として回線に送出されていた。これは電話機がどのようなハードウエアを用いて設計されるのかを知識を持つ設計者が分析を行なうと分析段階で発見可能であった。

前節では本研究で作成した分析モデルと、コードから作成した分析モデルの相違点を明らかにしたが、分析モデルの修正はインタラクション図にも大きな影響を及ぼす。よってコードから抽

出したインタラクション図を示し、以前に作成したインタラクション図との比較を行なう。図 4.2

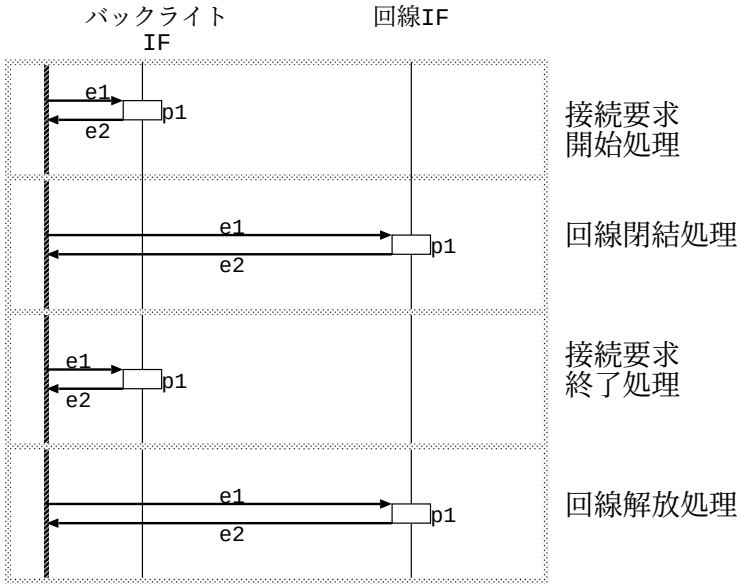


図 4.2: 着信処理ユースケースに関するインタラクション 1

はバックライト IF 、回線 IF がユースケースハンドラ、回線ハンドラとどのように通信するかに注目して設計した図である。

- 接続要求開始処理
  - p1：バックライト ON
    バックライト IF においてバックライトを点灯させる処理である。
  - e1：バックライト ON 要求
  - e2：バックライト ON 要求終了通知
    この処理はユースケースハンドラまたは回線ハンドラによって送られるイベント e1 によって実行が開始され処理の終了はイベント e2 を送ることで伝えられる。
- 回線閉結処理
  - p1：回線閉結処理
    は回線 IF において実行される回線閉結処理を表す。
  - e1：回線閉結処理要求
  - e2：回線閉結処理要求終了通知
    これは電話利用者ハンドラから送られるイベント e1 によって実行が開始され処理の終

了をイベント e2 を電話利用者ハンドラに返すことで伝える。

- 接続要求終了処理
  - p1：バックライト OFF
    はバックライト IF において実行される処理でバックライトを OFF にする。
  - e1：バックライト OFF 要求
  - e2：バックライト OFF 要求終了通知
    これはユースケースハンドラによって送信されるイベント e1 によって開始され処理の終了をイベント e2 をユースケースハンドラに送ることで伝えている。
- 回線開放処理
  - p1：回線開放処理
    は回線 IF によって実行される処理で回線開放処理を行なう。
  - e1：回線開放処理要求
  - e2：回線開放処理終了通知
    これは電話利用者ハンドラから送られるイベント e1 によって処理が開始され、処理の終了をイベント e2 を電話利用者ハンドラに送ることによって知らせている。

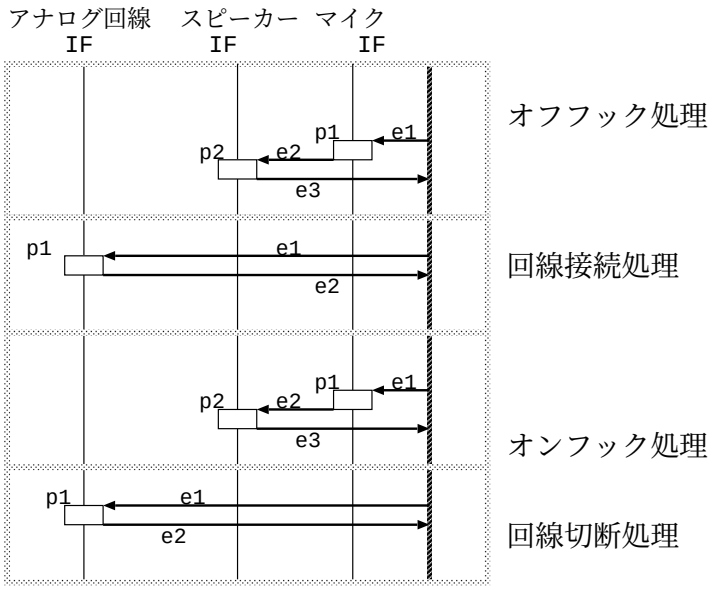


図 4.3: 着信処理ユースケースに関するインタラクション 2

図 4.3は電話利用者ハンドラ、ユースケースハンドラ、アナログ回線ハンドラが、マイク IF、スピーカー IF、とどのように通信を行なうかに注目して設計した図である。

- オフフック処理
  - p1：マイク ON
    マイク IF によって実行されるマイク ON という処理を表す。
  - e1：マイク ON 要求
  - e2：マイク ON 要求終了通知
    これは電話利用者ハンドラによって送られる e1 というイベントによって起動され、e2 というイベントをスピーカー IF に送ることで終了を通知している。スピーカー IF は e2 というイベントを受け取りスピーカーを ON にする。
  - p2：スピーカー ON
    スピーカー IF によって実行されるスピーカー ON という処理を表す。
  - e3：スピーカー ON 要求終了通知
    e3 というイベントを電話利用者ハンドラに送ることで終了を伝える。
  - p1：回線接続処理アナログ回線 IF において実行される処理でマイク、スピーカーと回線の接続処理を行なう。

- アナログ回線 IF は e1 というイベントを受け取り、回線接続処理を行なう。
- e2：回線接続処理要求終了通知
  これはスピーカー IF から送られるイベント e1 によって開始され、処理の終了をイベント e2 を送ることで伝えている。

- オンフック処理
  - p1：マイク OFF
    マイク IF で実行される処理でマイク OFF を表す。
  - e1：マイク OFF 処理要求
  - e2：マイク OFF 処理終了通知
    電話利用者ハンドラによって送信されるイベント e1 によって起動される処理で、e2 というイベントをスピーカー IF に送ることで処理の終了を知らせる。スピーカー IF は e2 というイベントを受け取り、スピーカーを OFF にする。
  - p2：スピーカー OFF
    スピーカー IF で実行される処理でスピーカー OFF を表す。
  - e3：スピーカー OFF 要求終了通知
    e3 というイベントを電話利用者ハンドラに送ることで処理の終了を知らせる。

  - p1：回線切断処理
    アナログ回線 IF において実行される処理で、マイク、スピーカーと回線の切断処理を行なう。
  - アナログ回線 IF は e1 というイベントを受けとり回線切断処理を開始する。
  - e2：回線切断処理要求終了通知
    スピーカー IF から送られるイベント e1 によって開始され、処理の終了をイベント e1 を送ることで伝えている。

図 4.4は回線ハンドラがベル IF、回線 IF、バックライト IF 、ユースケースハンドラとどのように通信を行なうのかに注目して設計した図である。

- 回線閉結処理
  - p1：回線閉結処理
    回線 IF において実行される、回線閉結処理を表す。
  - e1：回線閉結処理要求

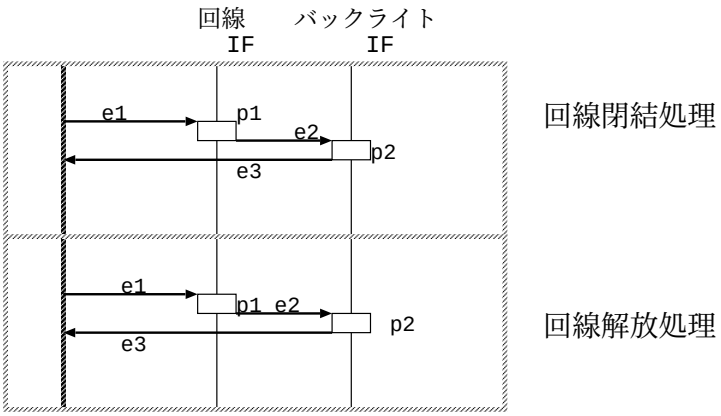


図 4.4: 発信処理ユースケースに関するインタラクション 1

- e2：回線閉結要求終了通知
  これはユースケースハンドラから送られる e1 というイベントによって起動し、e2 というイベントをバックライト IF に送ることで処理の終了を伝える。バックライト IF はイベント e2 を受け取り処理を開始する。
- p2：ライト ON 処理
  バックライト IF において実行される、ライトの ON という処理を表す。
- e3：バックライト ON 処理要求終了通知
  回線 IF から送られる e2 というイベントを受け取り、バックライト ON 処理を行ない、処理の終了を e3 というイベントを電話利用者ハンドラに送ることで伝える。

- 回線開放処理
  - p1：回線開放処理
    回線 IF において実行される回線開放処理を表す。
  - e1：回線開放処理要求
  - e2：回線開放処理要求終了通知
    これはユースケースハンドラによって送信されるイベント e1 によって起動される。イベント e2 の送信によって終了をバックライト IF に知らせる。バックライト IF はイベント e2 を受け取ることによってバックライト OFF 処理を行なう。
  - p2：バックライト OFF
    バックライト IF において実行されるバックライト OFF という処理を表す。

– e3：バックライト OFF 処理要求終了通知
　e3 というイベントを電話利用者ハンドラに送ることで処理の終了を通知する。

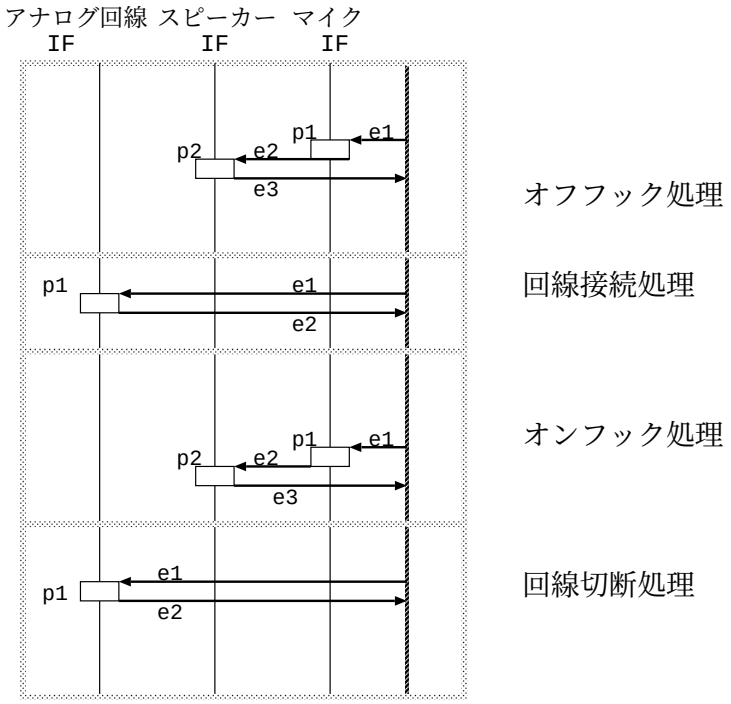


図 4.5: 発信処理に関するインタラクション 2

図 4.5は発信処理ユースケースについて、ユースケースハンドラ、利用者ハンドラ、アナログ回線ハンドラが他のインターフェイスオブジェクトとどのように通信するのかに着目して設計した図である。

- オフフック処理
  - p1：マイク ON
    マイク IF によって実行されるマイク ON という処理を表す。
  - e1：マイク ON 要求
  - e2：マイク ON 要求終了通知
    これは電話利用者ハンドラによって送られる e1 というイベントによって起動され、e2 というイベントをスピーカー IF に送ることで終了を通知している。スピーカー IF は e2 というイベントを受け取りスピーカーを ON にする。

- p2：スピーカー ON
  スピーカー IF によって実行されるスピーカー ON という処理を表す。
- e3：スピーカー ON 要求終了通知
  e3 というイベントを電話利用者ハンドラに送ることで終了を伝える。

- p1：回線接続処理アナログ回線 IF において実行される処理でマイク、スピーカーと回線の接続処理を行なう。
- アナログ回線 IF は e1 というイベントを受け取り、回線接続処理を行なう。
- e2：回線接続処理要求終了通知
  これはスピーカー IF から送られるイベント e1 によって開始され、処理の終了をイベント e2 を送ることで伝えている。

- オンフック処理

  - p1：マイク OFF
    マイク IF で実行される処理でマイク OFF を表す。
  - e1：マイク OFF 処理要求
  - e2：マイク OFF 処理終了通知
    電話利用者ハンドラによって送信されるイベント e1 によって起動される処理で、e2 というイベントをスピーカー IF に送ることで処理の終了を知らせる。スピーカー IF は e2 というイベントを受け取り、スピーカーを OFF にする。
  - p2：スピーカー OFF
    スピーカー IF で実行される処理でスピーカー OFF を表す。
  - e3：スピーカー OFF 要求終了通知
    e3 というイベントを電話利用者ハンドラに送ることで処理の終了を知らせる。

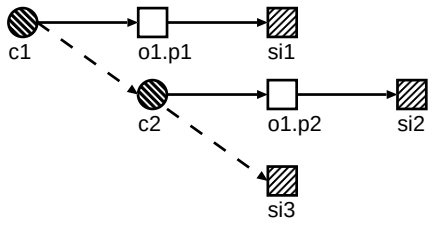
  - p1：回線切断処理
    アナログ回線 IF において実行される処理で、マイク、スピーカーと回線の切断処理を行なう。
  - アナログ回線 IF は e1 というイベントを受けとり回線切断処理を開始する。
  - e2：回線切断処理要求終了通知
    スピーカー IF から送られるイベント e1 によって開始され、処理の終了をイベント e1 を送ることで伝えている。

4.2 SES の抽出

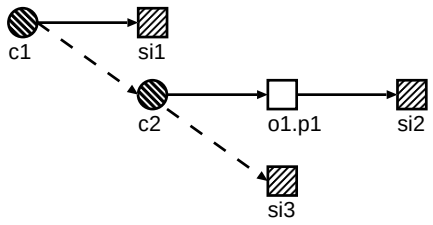
ここではまずコードから SES を定義に従って抽出する。また SES 抽出の際に SES を識別するため、番号付けを行なう。さらにオブジェクトと、オブジェクトの処理を明確にするために以下のように識別子と対応させる。

- o1：バックライト IF
- o2：マイク IF
- o3：スピーカー IF
- o4：外部回線 IF
- o5：内部 (アナログ) 回線 IF

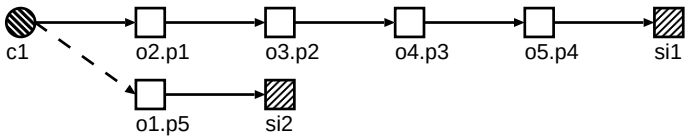
条件分岐については、図中実線で示される分岐がデフォルトの分岐。破線で示されているのが例外処理に移る分岐であることを意味する。シグナル si は実行中の SES から、他の SES に移るときに発行される。またシグナルには特別なシグナルとして、自己遷移のためのシグナルと、初期状態の SES へ移るためのシグナルがある。



- o1.p2：バックライト ON/OFF。電話利機に付属するバックライトを ON にするという記述が書かれている。
- si2：オフフックを知らせ他の SES へ移るためのシグナル
- si3：自己遷移するためのシグナル



- c1：オフフック状態に関する条件分岐
  オンフックされているとデフォルト。
- o1.p1：バックライト OFF
  電話機に付属するバックライトを OFF にするための処理が記述されている。
- si1：初期状態へ移るためのシグナル
- si2：自己遷移するためのシグナル



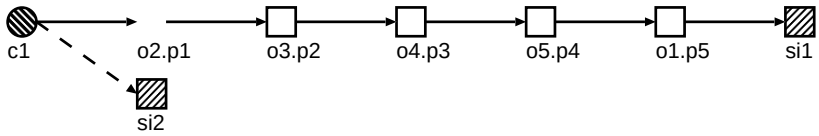
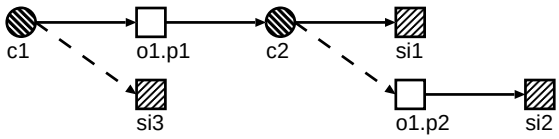


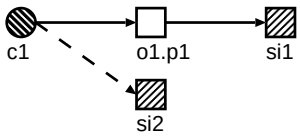
図 4.10: SES 5

図 4.10で示されている処理について説明する。

- c1：フック状態に関する条件分岐
  オンフックされていればデフォルト。
- o2.p1：マイク OFF
  受話器に付属するマイクを OFF にする処理が記述されている。
- o3.p2：スピーカー OFF
  受話器に付属するスピーカーを OFF にするための処理が記述されている。
- o4.p3：回線開放
  回線を外部に開放するための処理が記述されている。
- o5.p4：内部回線切断
  内部回線とマイク、スピーカーを切断するための処理が記述されている。
- si1：オンフックを知らせ他の SES 移るためのシグナル。
- o1.p5：バックライト OFF
  電話機に付属するバックライトを OFF にするための処理が記述されている。
- si2：初期状態の SES に移るためのシグナル



- c1：回線接続要求に関する条件分岐
  回線接続要求が来ていればデフォルト分岐
- o1.p1：バックライト ON
  電話機に付属するバックライトを ON にするための処理が記述されている。
- c2：フック状態に関する条件分岐
  オフフックされていればデフォルト分岐。
- si1：オフフックを知らせ他の SES へ移るためのシグナル
- o1.p2：バックライト OFF。
  電話機に付属するバックライトを OFF にするための処理が記述されている。
- si2：初期状態に移るためのシグナル
- si3：自己遷移のためのシグナル



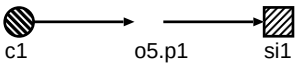
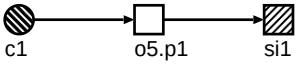


図 4.13: SES 8



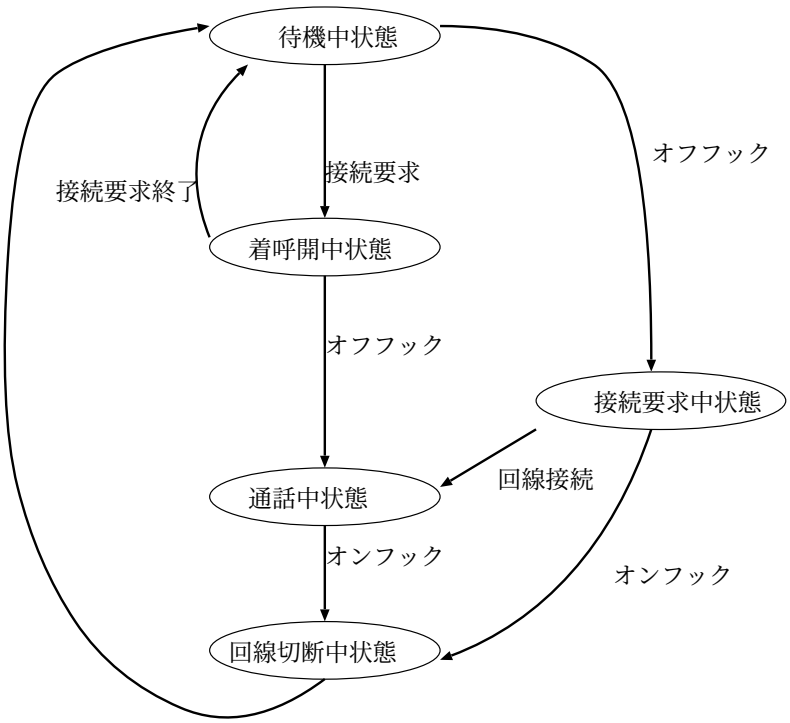


図 4.15: SES-Based 設計モデルにおける状態遷移図

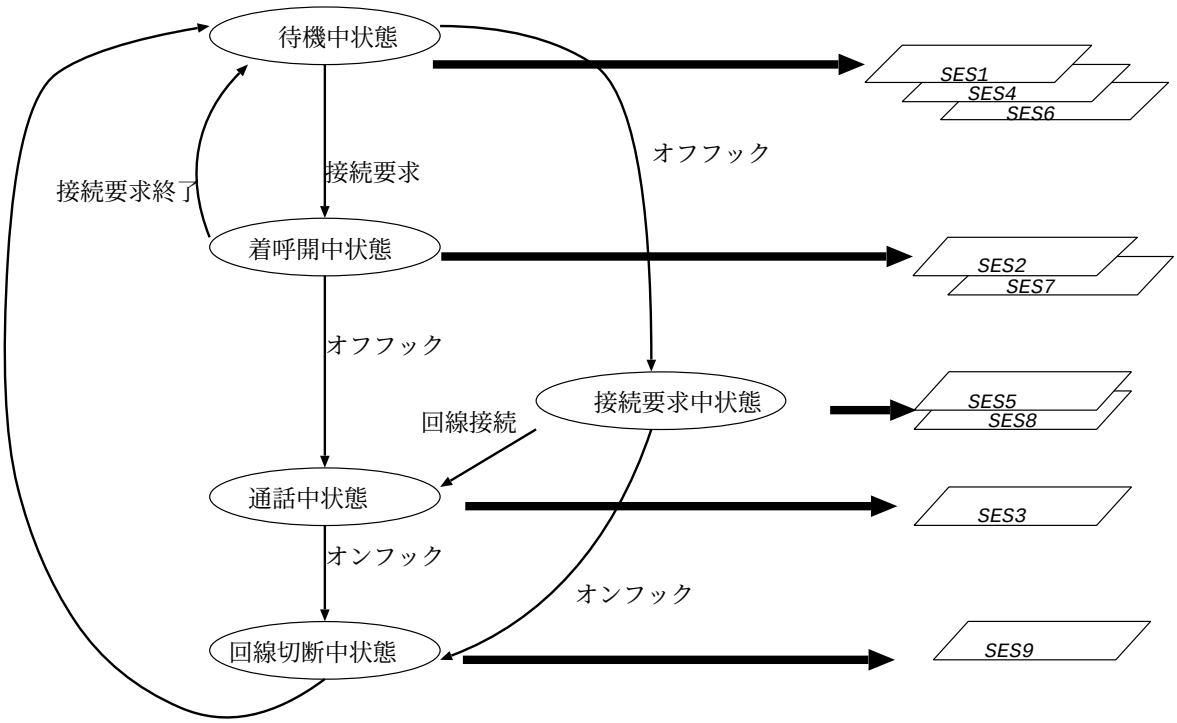


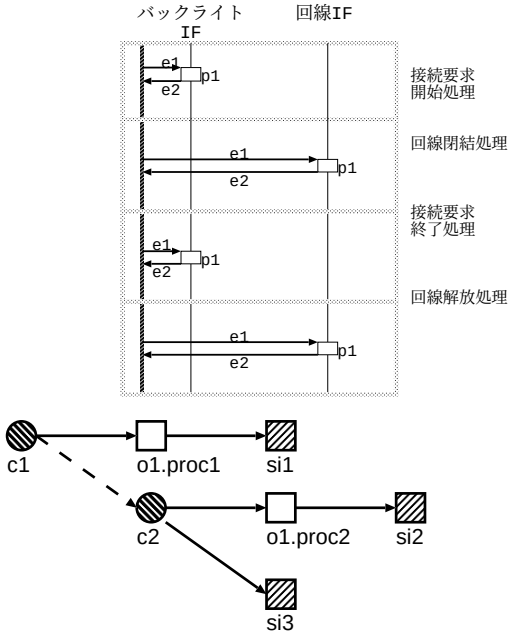
図 4.16: SES-Based 設計モデル

第 5 章

考察

5.1 SES とインタラクション図の対応

抽出した SES とインタラクション図の対応を調べる。まずそのために SES においてプロセスの表記を proc と改め、インタラクション図におけるプロセスの表記 p との衝突をさける。図 5.1で



1. c1：接続要求に関する条件分岐
   回線からの接続要求が届いていればデフォルトの処理。この例ではデフォルトの処理列が実行される。

2. o1.proc1：バックライト ON
   電話機に付属するバックライトを ON にする。

3. si1：接続要求を知らせ他の SES へ移るためのシグナル

これはインタラクション図において、接続要求開始処理に含まれる処理 p1 (バックライト ON) と対応する。

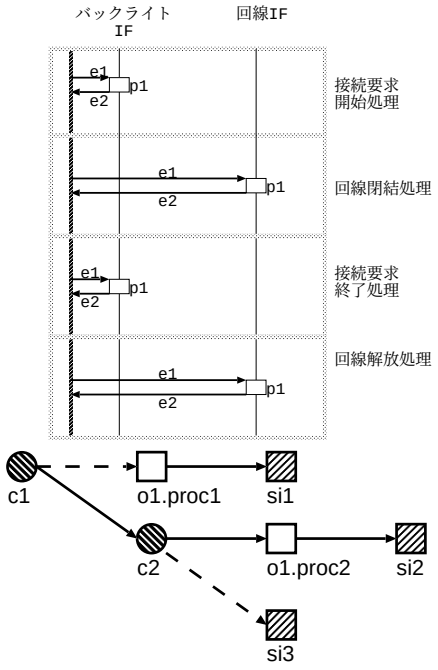


図 5.2: SES との対応 2

図 5.2における SES の処理を示す。

1. c1：接続要求に関する条件分岐
   回線からの接続要求が届いていればデフォルトの処理。この例では例外となる処理列が実行される。

2. c2：オフフックされているかについての条件分岐。
   この例ではオフフックされデフォルトの処理が実行される。

3. o1.proc2：バックライト ON/OFF。電話利機に付属するバックライトを ON にするという記述が書かれている。

4. si2：オフフックを知らせ他の SES へ移るためのシグナル

これはインタラクション図において接続要求開始処理に含まれる、p1(バックライト ON) に対応する。

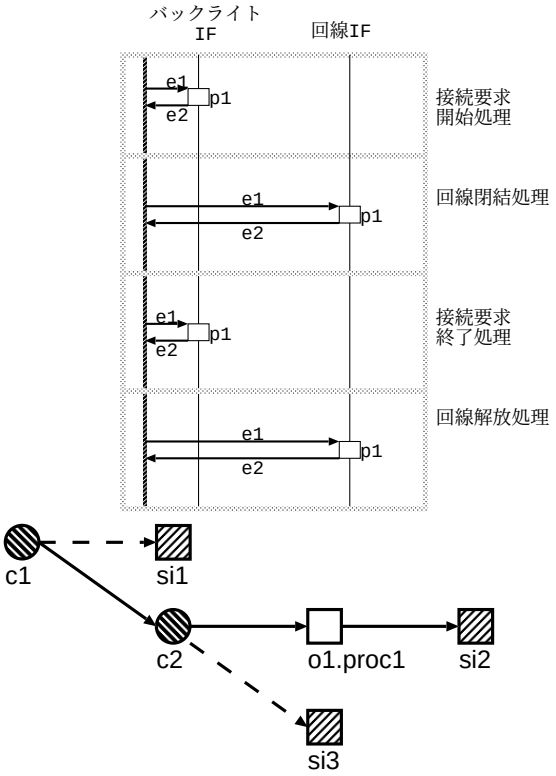


図 5.3: SES との対応３

図 5.3において SES で行なわれている処理を次に示す。

1. c1：フック状態に関する条件分岐
   オンフックしていればデフォルトの処理。この例では例外処理が実行される。

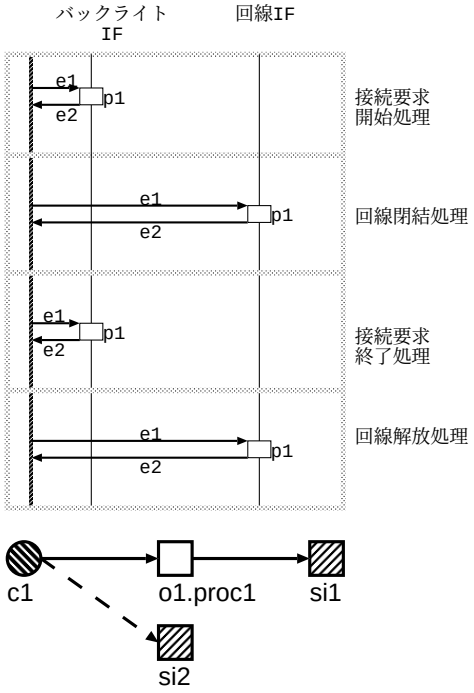
2. si1：オンフックを知らせ他の SES へ移るためのシグナル

3. c2：回線から接続要求終了に関する条件分岐
   回線接続要求が終了していればデフォルトの処理。ここではデフォルトの処理が行なわれている。

4. o1.proc1：バックライト OFF
   電話機に付属するバックライトを OFF にする処理が記述されている。

5. si2：初期状態にある SES へ移るためのシグナル

これはインタラクション図において接続要求終了処理に含まれる、p1(バックライト OFF) と対応する。



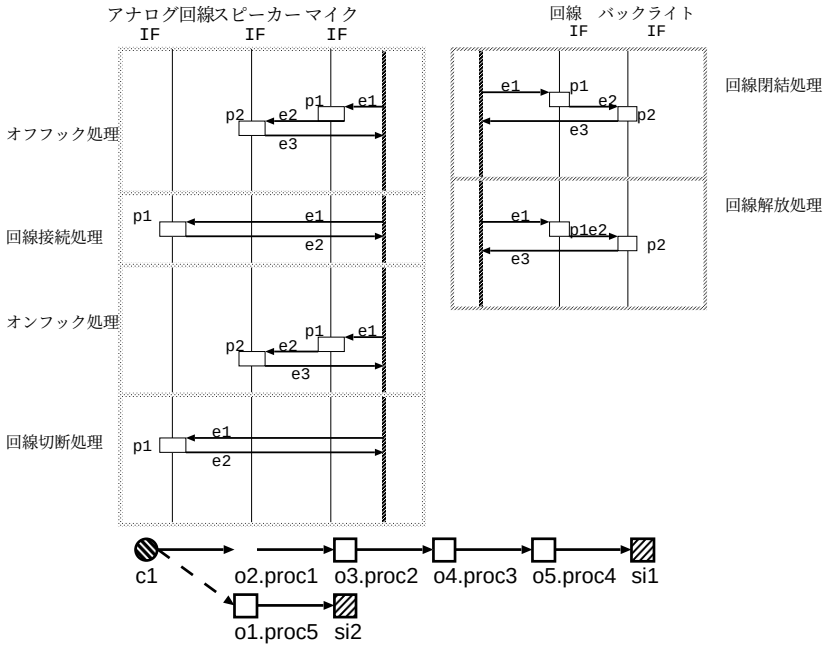


図 5.5: SES との対応５

1. c1 ではオフフックしているかどうかの条件分岐である。
   オフフックしていればデフォルトの処理が実行される。ここで挙げる例の場合、オフフックされていて次に p1 に処理が移っている。

2. o2.proc1:マイクを ON にするという処理が記述されている。

3. o3.proc2:スピーカーを ON にするという処理が記述されている。

4. o4.proc3:回線閉結を行なう処理が記述されている。

5. o5.proc4:回線とマイク、スピーカーの接続処理が記述されている。

6. si1：オフフックを知らせ他の SES 移るためのシグナル

これはインタラクション図では、オフフック処理における処理 p1、p2(マイク ON、スピーカー ON) 回線接続処理における処理 p1(アナログ回線接続) と回線閉結処理における p2 という処理に対応する。

インタラクション図から SES への変換を考えた場合、設計段階で発見される特性を考慮して SES への変換を考える必要がある。この例では、インタラクション図で表現された複数の処理を、処理列にまとめて逐次に行なうほうが効率が良いと判断して設計したと考えられる。これは処理時間、あるいはメモリ効率などといった特性評価からの判断と考えられる。図 5.6に表されている

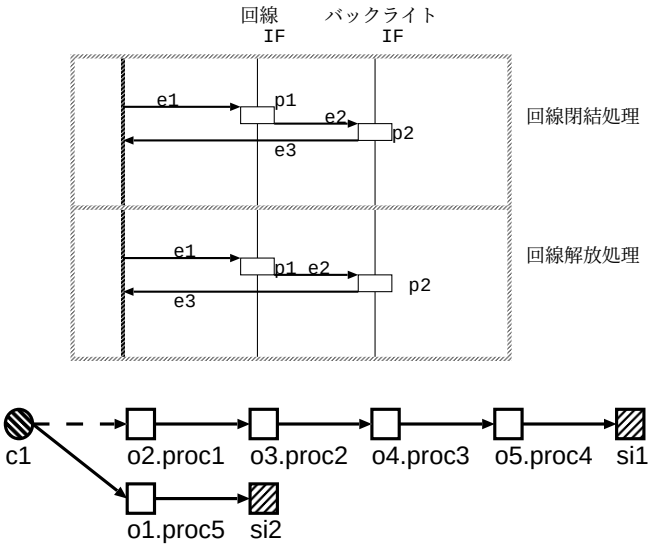


図 5.6: SES との対応６

SES の処理を次に示す。

1. c1：フック状態に関する条件分岐
   オフフックされていればデフォルトの処理が実行される。この例では例外となる処理が実行される。

2. o1.proc5：バックライト OFF
   電話機に付属するバックライトを OFF にするための処理が記述されている。

3. si2：オンフックを知らせ他の SES 移るためのシグナル

これはインタラクション図における、回線開放処理に含まれる処理 p2(バックライト OFF) に対応する。

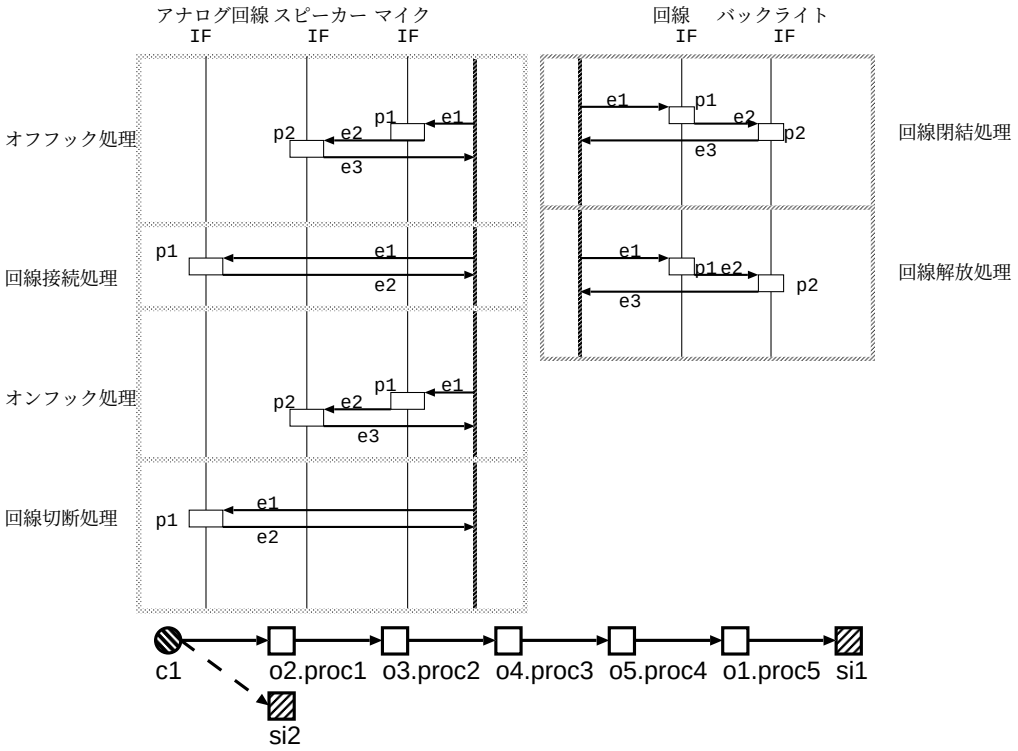


図 5.7: SES との対応 7

図 5.7における SES の処理を次に示す。

1. c1：フック状態に関する条件分岐
   オンフックされていればデフォルトの処理が実行される。この例ではデフォルトの処理が実行される。

2. o2.proc1：マイク OFF
   受話器に付属するマイクを OFF にする処理が記述されている。

3. o3.proc2：スピーカー OFF
   受話器に付属するスピーカーを OFF にするための処理が記述されている。

4. o4.proc3：回線開放
   回線を外部に開放するための処理が記述されている。

5. o5.proc4：内部回線切断
   内部回線とマイク、スピーカーを切断するための処理が記述されている。

6. si1：オンフックを知らせ他の SES 移るためのシグナル。

これはインタラクション図ではオフフック処理に含まれる p1,p2(マイク OFF、スピーカー OFF)、回線切断処理 p1(アナログ回線切断) と、回線開放処理に含まれる p2(バックライト OFF) に対応する。

これらの複数の処理が SES において一つの処理列にまとめられているのは、処理の間隔や処理時間が非常に短いためであったことが、実装段階において明らかになったためだと推測される。

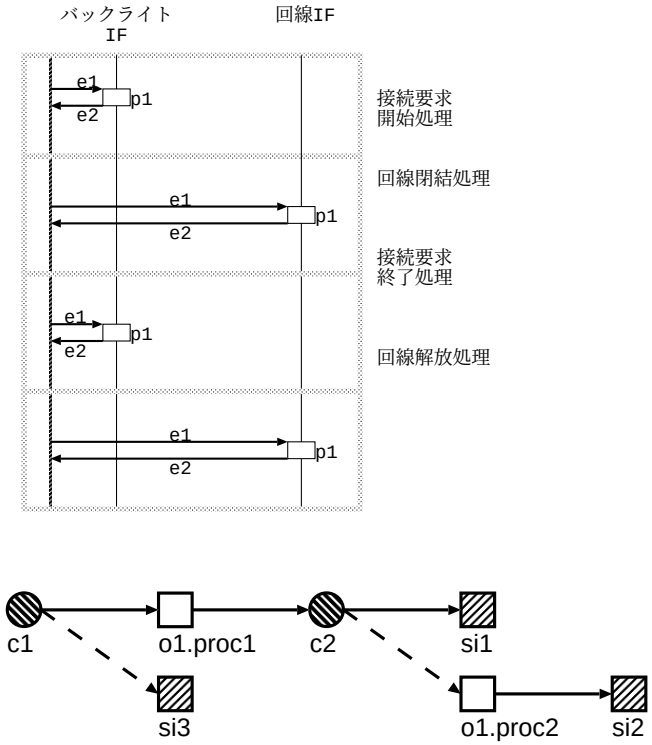


図 5.8: SES との対応８

図 5.8における SES の処理を次に示す。

1. c1：回線接続要求に関する条件分岐
   外部から回線接続要求が来ていればデフォルトの処理が実行される。この例ではデフォルトの処理が実行される。

2. o1.proc1：バックライト ON
   電話機に付属するバックライトを ON にするための処理が記述されている。

3. c2：フック状態に関する条件分岐
   オフフックされていればデフォルトの処理が実行される。この例はオフフックされてデフォルトの処理が実行されている。

4. si1：オフフックを知らせ他の SES へ移るためのシグナル

これはインタラクション図において接続要求開始処理に含まれる処理 p1(バックライト ON) に対応する。

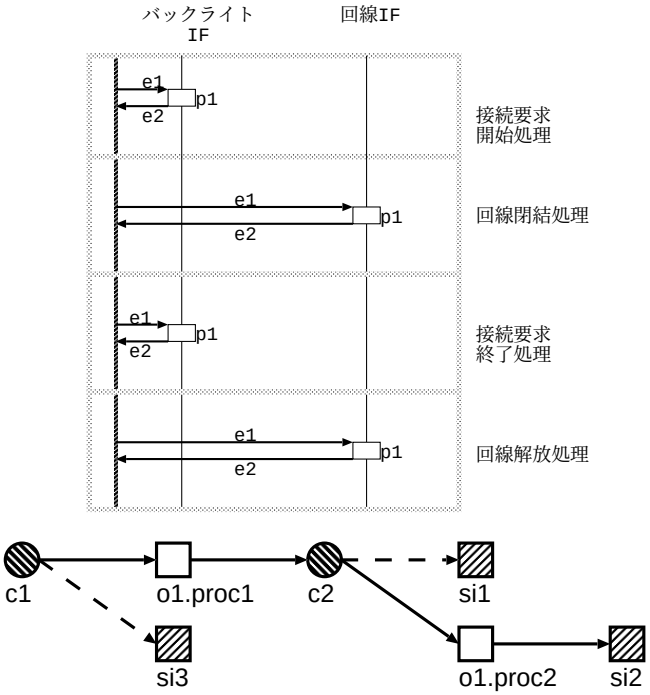


図 5.9: SES との対応 9

図 5.9における SES の処理を次に示す。

1. c1：回線接続要求に関する条件分岐
   外部から回線接続要求が来ていればデフォルトの処理が実行される。この例では例外となる処理が実行される。

2. o1.proc1：バックライト ON
   電話機に付属するバックライトを ON にするための処理が記述されている。

3. c2：フック状態に関する条件分岐
   オフフックされていればデフォルトの処理が実行される。この例では例外処理が実行される。

4. o1.proc2：バックライト OFF。
   電話機に付属するバックライトを OFF にするための処理が記述されている。

5. si2：初期状態に移るためのシグナル

これはインタラクション図において、接続要求開始処理の処理 p1(バックライト ON) と接続要求終了処理の処理 p1(バックライト OFF) に対応する。

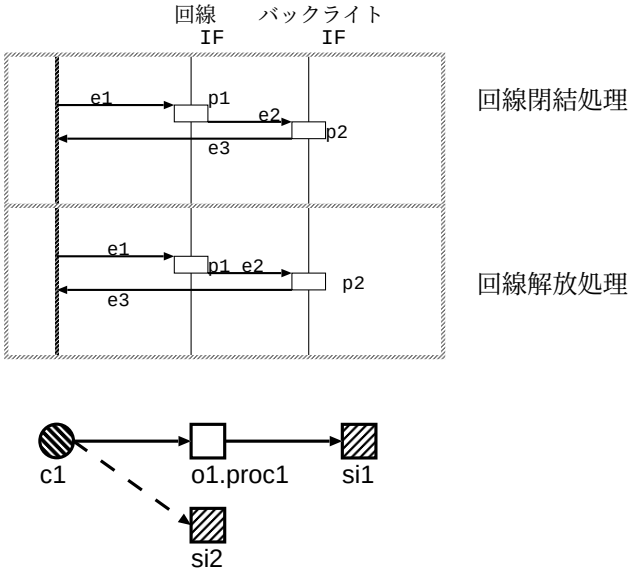


図 5.10: SES との対応１０

図 5.10における SES の処理を次に示す。

1. c1：フック状態についての条件分岐
   オフフックされていればデフォルトの処理が実行される。この例ではデフォルトの処理が実行される。

2. o1.proc1：バックライト OFF
   電話機に付属するバックライトを OFF にするための処理が記述されている。

3. si1：初期状態に移るためのシグナル

これはインタラクション図において回線開放処理に含まれる処理 p2(バックライト OFF) に対応する。

図 5.11において SES で行なわれる処理を次に示す。

1. o5.proc1：外部回線接続要求
   外部回線へのマイク、スピーカーの接続要求処理が記述されている。

2. si1：回線接続を知らせ他の SES に移るためのシグナル

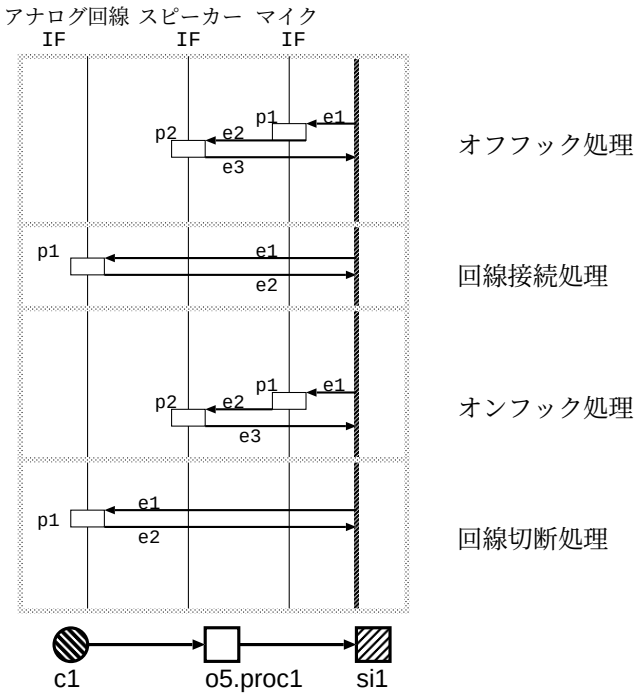


図 5.11: SES との対応１１

この処理はインタラクション図において回線接続処理に含まれる処理 p1(回線接続処理) に対応する。

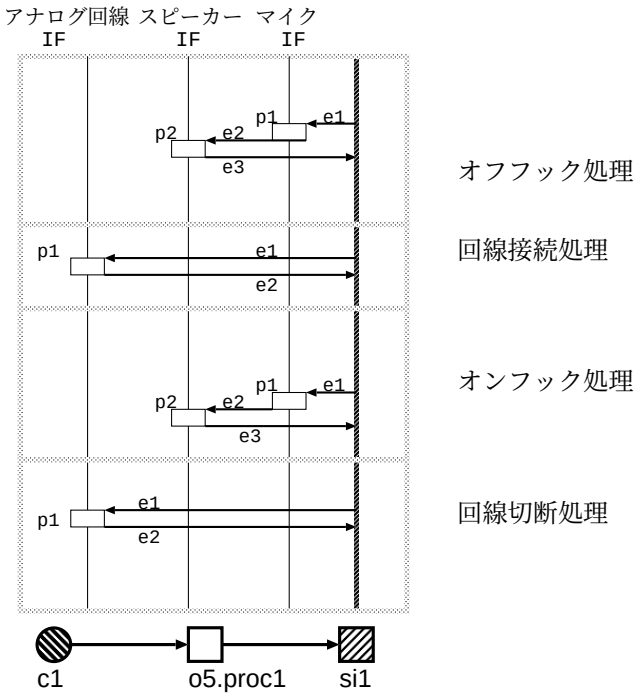


図 5.12: SES との対応１２

図 5.12における SES で行なわれる処理を次に示す。

1. o5.proc1：外部回線切断
   外部回線とマイク、スピーカーを切断する処理が記述されている。

2. si：回線切断を知らせ他の SES に移るためのシグナル

この処理はインタラクション図において回線切断処理に含まれる処理 p1(回線切断) に対応する。

5.2 SES-Based ソフトウエアアーキテクチャ

SES-Based ソフトウエアアーキテクチャは、SES-Based アプローチにおいて実装工程で実現される。ここでは考察としてこれまで得た成果物から SES-Based ソフトウエアアーキテクチャの全体像を示す。図 5.13において、Task において実行される SES は SES-Based 設計モデルから得る

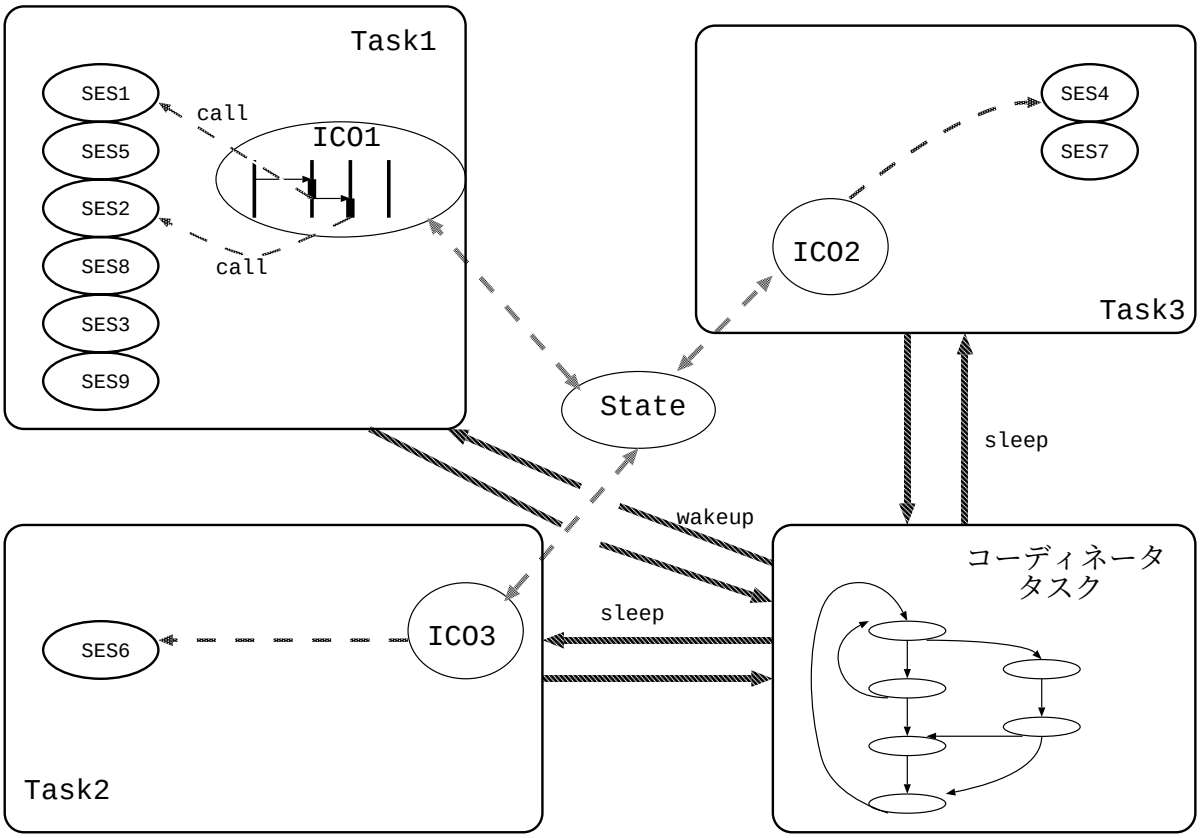


図 5.13: SES-Based ソフトウエアアーキテクチャの全体像

ことができる。また Task の中で ICO と記されているものは、SES の制御を行なうインプリシットオブジェクトを表す。ICO はその制御を決定するための情報を state から得る。state の値はこの例の場合、フック状態や着信状態などが用いられることが SES の条件分岐を調べることによって導かれる。コーディネータタスクについては、SES-Based 設計モデルから導出することが可能である。

5.3 SES-Based ソフトウエアアーキテクチャの評価

SES-Based アプローチおよび、SES-Based 設計モデルを以下の観点から評価する。

- 構造の明示的な表現。
  大規模なシステムである程その構造を正しく捉え、開発者相互で理解を共有することが難しくなる。SES-Based ソフトウエアアーキテクチャの意味構造を、開発時において理解することの容易性について考察する。

  – SES-Based アプローチは電話機の組み込みシステム開発という特定のドメインを仮定して提案されている。電話機の組み込みシステムは組み込みシステムの適用分野の中では、小〜中規模である。しかし実際に大規模システム開発に用いられるかどうかは、組み込みシステムの性質上実際にそのドメインの特性を調査した上でないと判断することは困難である。本研究事例で得た経験から推測されることは、システムの規模よりも、オブジェクト間のインタラクションの複雑さや、設計段階のモデル構築の際にシステムが取り得る状態の複雑さに SES-Based ソフトウエアアーキテクチャの理解の容易性は依存すると思われる。

- 非機能的特性との関係。
  組み込みシステムに限らず、システム開発において非機能的特性を実現することは重要な要素である。

  またシステムの、性能、信頼性、再利用性、などの非機能的特性はソフトウエアの構造によって特徴づけることができる。よってソフトウエアアーキテクチャとそれによって実現される非機能的特性との関係といった観点に注目し評価する。

  – ソフトウエアアーキテクチャと非機能的制約の関係という観点から述べると、SES-Based 設計モデル、SES-Based ソフトウエアアーキテクチャはその定義から、SES という処理時間情報の単位を用いて設計を行なうことで性能に関する非機能的性質のひとつである実時間性に関する性質を扱うことと可能にしている。

- 設計視点。
  ソフトウエアの複雑度が高まるにつれ、ソフトウエアの設計視点は単に機能面にとどまらず、多様な視点から設計を行なう必要が生じてきた。SES-Based 設計モデルがどのような視点から設計されているのかに注目し評価する。

  – SES-Based アプローチではオブジェクト指向方法論を基にシステムの分析を行なうことを仮定している。よってその分析工程では、ソフトウエアの動的、静的な側面 (視点) を考慮してシステム分析／設計が行なわれることがいえる。
    また SES-Based アプローチでは SES-Based 設計モデルの構築時において、インタラクション図から処理時間情報を扱いやすい性質を持つ SES を抽出し設計を行なうことによって、実時間性という非機能的な側面 (視点) からシステムの設計を行なっている。

- 再利用性
  ソフトウエアアーキテクチャはその定義から、ドメインに依存した構造を取らざるを得ない。しかしその依存度が高くなるに従って、ソフトウエア部品としての再利用度も低くなる。SES-Based アプローチを再利用性という観点に注目して評価を行う。

  – SES-Based アプローチは電話機の組み込みシステムというドメインを仮定して提案されている。しかし組み込みシステムというドメインの視点から考えた場合、SES-Based 設計モデル、SES-Based ソフトウエアアーキテクチャは実時間性を考慮しやすいアーキテクチャを持つといえる。

5.4 SES-Based アプローチに対する提案

組み込みシステム開発において、満たされなければならない非機能的制約に実時間性に対する制約、ハードウエアに関する制約が挙げられる。
SES-Based アプローチでは、時間量を扱いやすい構造 (SES-Based 設計モデル、SES-Based ソフトウエアアーキテクチャ) をシステムの設計段階に用いることによって実時間性に対する制約を考慮しやすいという特性を持つ。

そこで我々が提案した SES-Based 設計モデルにおいて、時間量に関する特徴と、タスク数 (メモリの使用量に関する特徴) について以下で定義を行い、その定義を用いた効率の良いシステム構築方法を考察する。

5.4.1 時間量に関する定義

- プロセス処理時間
  ses におけるプロセス pid の処理時間は以下に定義される
  Time_process:ProcessID $\rightarrow$ PositiveRealNumber
  $where$ PositiveRealNumber=$\{n \in R \mid n > 0\}$
  SES-Based 設計モデルでは以下の制約を満たしているものと仮定する。
  $\forall$ pid $\in$ SES
  Time_process(pid):$defined$

- ses 処理時間
  ses の処理時間は、各々の ses において最も時間のかかる処理列によって定義する。以下に ses の処理時間 Time_ses を定義する。

Time_ses(END) =0
Time_ses(NEXT(op*ses))
= Time_process(op) + Time_(ses)
Time_ses(BRANCH(condition,ses,ses))
=max{(Time_ses(ses1),Time_ses(ses2)}

- オペレーションの総処理時間：PROCESS_time
  PROCESS_time：$\sum_{pid} Time\_process(pid)$
  ここで定義されるオペレーションの総処理時間では、全てのsesでのプロセスタイムの合計が求められる。

- sesの総処理時間：SES_time
  SES_time:$\sum_{ses} Time\_ses(ses)$
  ここで定義されるsesの総処理時間では、全てのsesが一回ずつ実行されるときの最大実行時間の合計が求められる。

5.4.2 タスク数の定義

- タスク数：タスクは、SES-Based設計モデルの状態遷移図において、状態に付属するsesの各々が異なるタスクで実行されるように設計される。
  よってシステムで用いられるタスク数は、一つの状態に付属するsesの最大個に依存する。
  タスク数は以下に定義される。Task_num = max | M_sys(s) |
  $where$ s ∈ S_sys

5.4.3 時間量の決定

システム全体の振舞いは、sesに含まれるconditionをどのように設計するかに依存する。既に定義したPROCESS_timeで表現される時間の特徴量はconditinonの設定にかかわらず設計段階で構築したモデルの論理的な構造によって決定される。
SES_timeで表現される時間の特徴量はsesに含まれるconditionをどのように設定するかに依存した形で出現する。

5.4.4 ses の抽出方法

ses の抽出方法に関して、実際に例題を行うことによって判明した点を明らかにする。

システム構築の際、分析段階で構築したモデルの論理的構造への影響を最小に押えたままで、ses を抽出する方法を考慮する。

ses にはその定義より以下の制約が含まれる。

1. ses に含まれる処理列は逐次に実行される。(並列に実行されない)
2. ses に含まれる処理列は同期して実行される。(ハード ウエアなどの処理待ちのためにブロックしない)

上記の条件によりインタラクション図において並列に実行されている箇所を、逐次実行されるように以下の例のように変更する。

- ユースケースの中で処理が並列に振舞われる場合
  オブジェクト A とオブジェクト B が並行に実行されているとき、オブジェクト B の処理が長時間の処理を行う可能性がある場合以下の条件で考える。
  (長時間の処理とはハード ウエアでの実行待ちや、他のアクターからの入力イベント待ちなどの処理を指し、CPU が処理待ちをする状況を指す)

  1. プロセス間にイベントの依存関係のあるとき。
     A で実行されるプロセスの出すイベントを利用して B でプロセスが実行される場合、A、B で実行されるプロセスを 2 つに分解し、2 つの ses を抽出する。
  2. プロセス間にイベントの依存関係がないとき。
     1 の方法を用いるか、A のプロセス終了後の B のプロセスを 2 つに分解して 2 つの ses を抽出する。

  ses を抽出することによって、ハード ウエアで実行される処理をアクターの処理としてインタラクション図におけるシステム境界の外側に出すことができる。上記の例の場合オブジェクト B のプロセス P の処理時間は前述の定義より Time_process(P) であり、プロセス P を P'、P" に分解したとすると、ここでのオブジェクト B におけるプロセスの処理時間は
  Time_process(P)=Time_process(P')+Time_process(P")+α(待ち時間)
  となる。
  これによって、システム境界の外側でアクターが処理を行う間、CPU の待ち時間αを他の依存関係のない処理に割り当てることができる。
  よって ses を抽出することによって、改善される cpu の利用効率を以下で示す定義を用いる

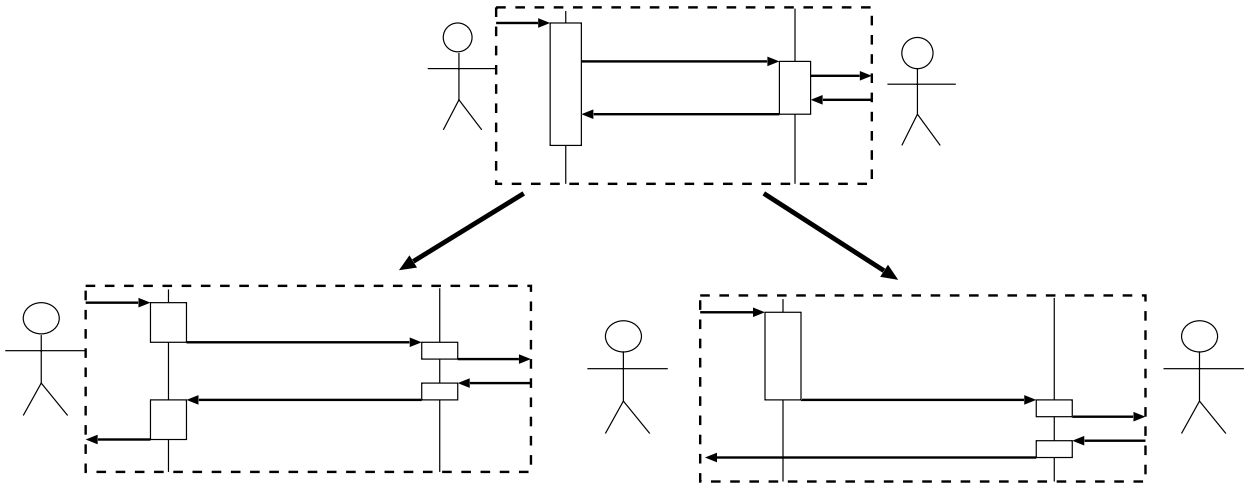


図 5.14: 並列実行からの ses 抽出

ことで示すことができる。
インタラクション図から ses を抽出するということは、プロセスをアクターの処理に変形し、そのインターフェイスとなる処理を新たに設けることを意味する。
そのため、ses の抽出前後でプロセスの処理時間は増加してしまう。
プロセスの処理時間の増加率はこれまでの定義された時間量を用いることにより定義することができる。ses の抽出後のプロセスの処理時間の増加率

$$P\_PROCTIME = \frac{Time\_ses(p') + Time\_ses(p'') + \alpha}{Time\_ses(p)}$$

ただしプロセス p を（p',p", アクターの処理）に分解したものとする。またαはアクターの処理時間である
次に ses を抽出したことによる cpu の可動時間の増加率を定義する。

$$P\_CPUTIME = \frac{SES\_time - \alpha}{SES\_time}$$

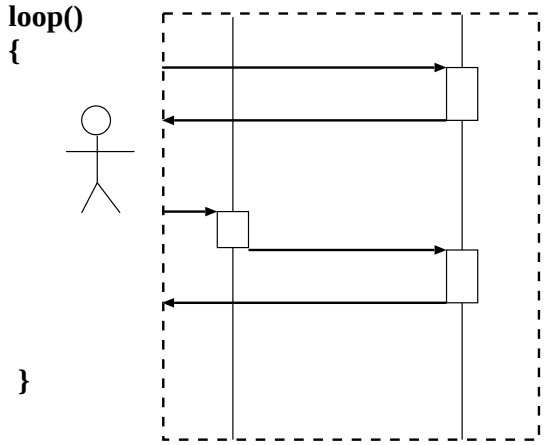
ｓｅｓを用いることによるシステムの利用効率はｓｅｓのプロセス処理時間の増加率と、ＣＰＵの可動時間の増加率との両方とを考慮しなければならない。そこでシステムの cpu 利用効率を以下で定義する。

$$utility = P\_PROCTIME * P\_CPUTIME$$

アクター抽出の条件

前節では ses 抽出の例を示したが、ses を抽出するためには、満たされなければならない条件がある。ハードウエアなどによる長時間の処理があった場合、ses を抽出することによって、抽出する ses の数が増えるとタスクの増加を招く。しかし長時間の処理を分解したことにより、プロセスの処理時間は以下で示す様に短縮される。よって両方の特徴を考慮してネゴシエーションを行う。

- ユースケースの中で処理が周期的に振舞う場合。
  周期的に振舞われるユースケースを一つの SES にまとめることは、いたづらにタスクが増加するのを防ぐことにつながる。しかしまとめられる処理が (同じ周期だが) 機能的に異なる場合、システムの構成が変更 (修正) されるとき同じ周期になるとは限らないことに注意する。SES が追加/削除されるとシステムの構成に大きな変更を伴う可能性が生じる。



べて非常に大きくなる可能性が多い。

5.4.5 ses の合成

これまでの方法で抽出した ses は状態に割り当てられるが、必ずしも一つの ses が一つの状態に対応するわけではない。
前述の様にハードウエアによる長時間の処理を、アクターに分ける場合は各々の ses は別の状態に配置される。
しかし、インタラクション図の分岐処理などから ses を抽出した場合、複数個の ses が同期して逐次実行する可能性がある。
このように複数個の ses が同期して逐次に実行される場合、それらの ses を一つの ses として状態に配置する必要がある。

第 6 章

まとめ

6.1 まとめ

本論文では組み込みシステム開発におけるソフトウエア構成法の調査として SES-Based アプローチを対象にして研究を行ない、その有効性について評価を行なった。

始めに SES-Based アプローチについて有効性を示すため、事例研究として電話機の組み込みシステム開発について SES-Based アプローチを適用した。次に事例研究の結果に基づき SES-Based アプローチの有効性を検証した。

またソフトウエアアーキテクチャの観点から、SES-Based アプローチを評価した。

6.2 今後の課題

- 本研究では、事例研究においてシステム開発において分析/設計工程まで研究を行なった。しかし SES-Based アプローチに基づいた完全な実装までには至っていない。よって SES-Based アプローチ基づいた実装工程までの検証を行なう。
- 組み込みシステムの特性である、非機能的要件について実装結果から評価を行ない SES-Based アプローチへの反映を行なう。

謝辞

本研究を行なうに当たり、御指導頂きました片山卓也教授では、多大な感謝を申し上げます。また、本研究に関して多くの助言を頂きました伊東恵助手、青木利晃氏ならびにソフトウエア基礎講座の皆様に厚く御礼申し上げます。

参考文献

[1] M.Shaw and D.Garlan: Software Architecture,Orentice Hall,1996.

[2] Jacobson,I.,Christerson, M., Jonsson, P., Overgaard, G.: Objsct-Oriented Software Engineering, Addison-Wesley, 1992.

[3] Ramgaugh,J.Blaha,M.,Premerlani,M.,Eddy,F. and Lorensen,W.: Obect-Oriented modeling and design, Prentice-Hall International, 1991.

[4] Toshiaki Aoki, Akira kawaguchi, Tomoji Kishi and Takuya Katayama: Synchronized Execution Sequence Based Software Architecture for Object-Oriented Embedded Systems, WICSA1,1999(to be presented)

[5] 川口晃, 岸知二, 門田浩:組み込みシステムの設計法-オブジェクト指向設計を中心にして-, 情報処理学会誌 Vol.38 No.10 Oct,1997

[6] 青木利晃, 内藤壮司, 片山卓也:オブジェクト指向組み込み開発のための SES-Based アプローチ, 第 15 回全国大会論文集, 日本ソフトウエア科学会, p.289-292,1998