修士論文

ＳＥＳアプローチに基づく実時間制約を考慮した組み込みシステムの実装法

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

森本 大喜

平成 12 年 2 月 15 日

目 次

第 1 章

はじめに

1.1　研究の背景

各種の機器に組み込まれてその制御を行うコンピュータシステムのことを、機器組み込み制御システム、あるいは単に組み込みシステムと呼ぶ。近年における半導体技術の発展に伴うマイクロプロセッサやメモリの低価格化により、組み込みシステムの応用分野は拡大の一途をたどっている。実際に、我々の身の回りに存在するほとんどの電子／電気機器に組み込みシステムが適用されるようになってきている。

現在、このようなハードウェア技術の進歩が制御対象となる機器の高機能化や複合化に拍車をかけ、組み込みシステムの規模を劇的に増大させている。そのため、システムの大規模化・複雑化に対して従来の開発手法による組み込みシステム開発が非常に困難なってきている。こういった状況では、システム全体の構造をどのように設計して実現するかが重要であり、この領域においてもオブジェクト指向開発手法が注目されている。しかし、組み込みシステム開発では、非機能的要件と呼ばれるこのドメイン特有の制約が存在する。この制約には実時間性、リアクティブ性、ハードウェア構成等などが挙げられる。実際の開発では非機能的要件による厳しい制約を考慮しなければならない。このため、分析工程中心の既存のオブジェクト指向開発手法を適応することが困難であり、これらの制約を扱う適切な設計手法を提案することが必要である。

本研究では、青木[1]により提案された SES アプローチに基づき、下流工程においてどのように非機能的要件を取り扱うべきかについて提案する。

1.2　目的

本研究では組み込みシステム開発手法である SES アプローチを対象に研究を行う。現状ではＳＥＳアプローチは実際のシステム開発に適用するまでには至っていない。そこで、電話機の組み込みシステム開発という特定のシステム開発に焦点を当て、これについて SES アプローチに基づく実装を行う。実装にはリアルタイム・オペレーティングシステム(RTOS ： Real Time Operating system)を用いている。RTOS はリアルタイム・システムを実現するための機能を備えたオペレーティングシステムである。一般に RTOS は組み込みシステムをターゲットとしており、メモリの使用量やサポートする CPU が組み込みシステム向けになっているものが多い。このため、実際の組み込みシステム開発現場では RTOS が多用されている。このような背景により、本研究における電話機を対象とした組み込みシステムの実装は RTOS を用いて行っている。

また実装ではアスペクト指向プログラミング(AOP ： Aspect-Oriented Programming)[2]に着目し、設計モデルからソフトウェアを獲得するまでの工程がどのように整理されるべきかを調査する。AOP では、コンポーネントプログラムでは記述しづらい側面をアスペクトプログラムとして記述し、アスペクトプログラムを Weave することによって最終的なソフトウェアを得る。

SES アプローチでは SES と呼ばれるものを処理の単位としており、RTOS ではタスクを処理の単位としている。そのため、まず SES をどのようにタスクにマッピングするかが問題となる。また、マッピング後のタスクのスケジューリング方法、さらにタスクにマッピングした SES が他の SES と通信を行うための方法を考える必要がある。よって、これらの実現方法に関する問題が浮上する。そこで、本研究では SES のタスクへの詰め込み方、タスクのスケジューリング、タスク間通信の３点をアスペクトとして捉え、AOP で実現している。さらに実装した結果から、これら３つが実時間制約に対してどのように影響し合うのかを調べ、SES アプローチに基づく実時間制約のある組み込みシステムの実装法について考察する。

1.3　構成

以下に本論文における構成を示す。

第１章　本論文の概要を説明する。

第２章　本研究で用いるSESアプローチについて説明する。

第３章　実装対象としている電話機システムの設計を行い、そこからSESを抽出する。さらに、抽出されたSESからSESモデルを構築する。

第４章　SESのタスクへの詰め込み方、タスクのスケジューリング、タスク間通信の３点をアスペクトとして捉え、これらに着目した、AOPによる電話機システムの実装法について説明する。

第５章　実装したものからアスペクトとして着目したものが、どのような関係があり、影響し合うのかを考察する。

第６章　本論文のまとめを行い、今後の課題を示す。

第２章

SES アプローチ

2.1　SES アプローチの概要

本研究の組み込みシステム開発におけるソフトウェア実装法の研究対象となるSES アプローチについて述べる。

SES アプローチはオブジェクト指向組み込みシステム開発手法であり、このアプローチでは SES(synchronized execution sequence)と呼ばれる同期した処理列を単位としている。この SES は時間の見積もりが難しいハードウェア処理による待ち時間や資源の競合によるブロックといった処理を含まないものである。そのため処理時間情報の単位として扱うことができ、実時間性を考慮しやすい設計の単位となっている。図 2.1 に示すように、個々の処理はオブジェクトのメソッドであり、SES に含まれる処理列はすべて短時間のうちに連続して実行される。

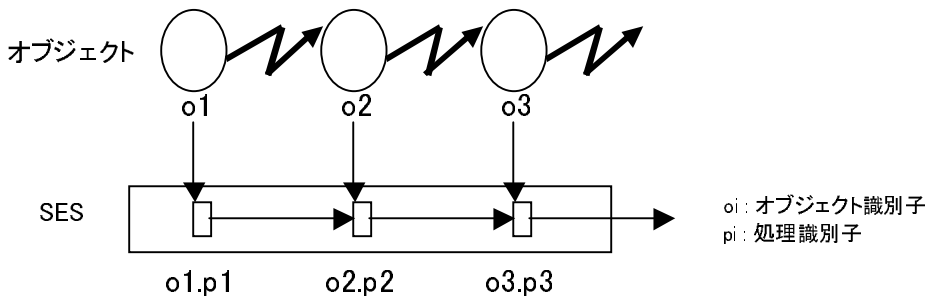


図 2.1　SES とオブジェクトの関係

また、図 2.2 に示すように、SES と状態遷移図から SES モデルが定義される。状態遷移図はシステム全体の全域的な状態と、それらの間の遷移関係により構成されるものである。状態遷移図の各状態に SES の集合が関連づけられており、SES の実行順序を整理してシステム全体の振る舞いを定義するモデルである。

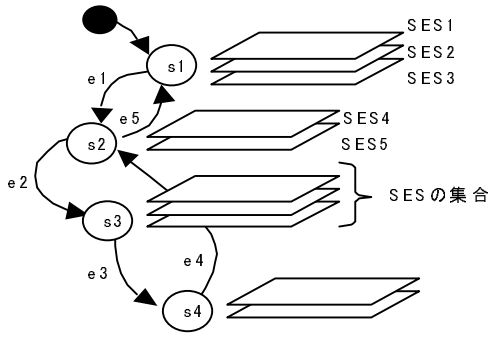


図 2.2　SES モデル

次に、図 2.3 のような SES モデルを挙げ SES の特徴について説明する。このモデルには状態 s1、s2 を持つ状態遷移図と、５つの SES（SES1～SES5）から構成している。ここでは、s1 を初期状態としている。また、状態 s1 には SES1 と SES2 が、状態 s2 には SES3、SES4、SES5 が付随している。

各状態の SES は並行に実行され、個々の SES はイベントが出力されるまで繰り返し実行される。SES は、オブジェクトが行う処理を一連に並べた処理列である。ここではオブジェクトを o、さらにその処理を p として表わしている。SES1 は o.p1、o.p2、o.p3 の３つの処理からなる処理列であり、o.p1 から順に実行されていく。また SES2 では o.p4 の処理を先頭に持ち、イベント e1 を処理列の最後に出力する。そして、このイベントにより状態遷移が起こる。このように、SES の最後の要素としてイベントを記述することができ、イベントは状態遷移のトリガーとして用いられる。

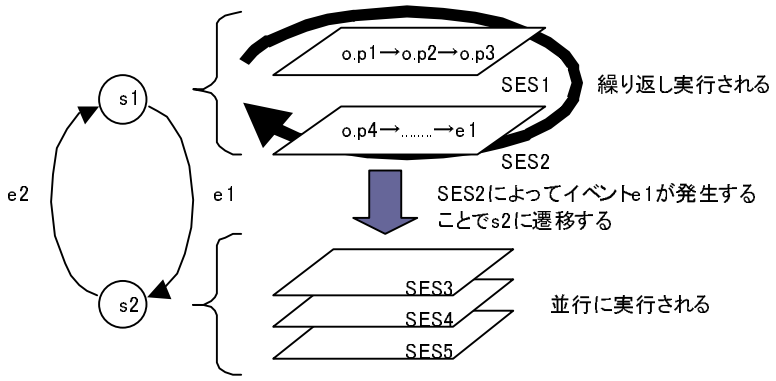


図 2.3　SES モデルの例

2.2　SES 処理列の表記法

SES はオブジェクトが行う処理を一連に並べた処理列であり、この処理列を構成する各々の処理には 4 種類挙げられる。上の図 2.3 では SES の処理列を o.p1→o.p2→o.p3 というように表しているが、この表記では実際にこれら 4 種類のどの処理にあたるのかを区別することはできない。そのため、SES の処理列がどのような処理により構成されているのかを明確にするため以下のような表記法を定義する。また、この表記法は 3 章において、本研究の対象である電話機システムから抽出された SES の処理列を表記するために用いる。

●　：条件分岐処理
　—実線で示されるのがデフォルトの分岐
　—破線で示されるのが例外処理に移る分岐
　　この表記の下には oX.cY と記述する。
　　o はオブジェクトを、c はオブジェクトのメソッドを意味し、X は処理を識別するための数値を割り当てる。

□　：条件分岐せず、単に処理列の 1 つとして存在する処理
　この表記の下には oX.pY と記述する。
　o はオブジェクトを、p はオブジェクトのメソッドを意味し、X は処理を識別するための数値を割り当てる。

■　：イベントを出力

この処理はｅで表す。

この表記の下にはeXと記述する。ｅはイベント出力を意味し、Xはイベントを出力する処理を識別するための数値を割り当てる。

■　：先頭の処理に戻る

この処理はｂで表す。

この表記の下にはbXと記述する。ｂは先頭処理に戻ることを意味し、Xには処理を識別する数値を割り当てる。

この表記法で処理列を表したSESの例を図2.4に示し、処理列を構成している個々の処理について説明する。

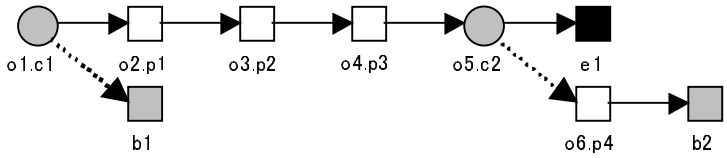


図2.4　表記法を用いたSESの処理列

上の図に示す各処理は矢印に従って順番に実行される。このSESでは条件分岐処理(o1.c1)が始めに処理される。ここで例外処理に移った場合、先頭に戻る処理（b1）によって再び条件分岐処理(o1.c1)が処理される。例外処理に移らずデフォルトの分岐に移った場合は、

o2.p1、o3.p2、o4.p3と３つのそれぞれ異なるオブジェクトの処理が連続してなされ、その後、条件分岐処理(o5.c2)が処理される。ここで例外処理に移った場合、o6.p4の処理がなされ、先頭に戻る処理(b2)により再び条件分岐処理(o1.c1)が処理されることなにる。またデフォルトの分岐に移った場合は、イベント(e1)が出力され他の状態へ遷移する。

第３章

電話機システムの設計

３．１　実装対象

本研究ではSESアプローチに基づいて実装を行うにあたり、電話機の組み込みシステム開発という特定のシステムを対象とする。電話機をその対象として取り扱ったのは、ＮＥＣマイコンソフト開発環境研究所の協力を得られたためである。そのため、実際に製品化されたコードレスホンシステムの開発事例[4]に関する資料を参考にしている。また、電話機を実装するための要求仕様の作成には製品化されたコードレスホンのユーザーズマニュアルとその他の資料を参考にしている。実際に製品化されたコードレスホンシステムの一部の機能について、SESモデルを作成し、RTOS上にプロトタイプの実装を行った。これより、その実装対象である電話機システムの設計過程を示す。

３．２　電話機の概要

実装の対象としている電話機を図 3.1 に示す。

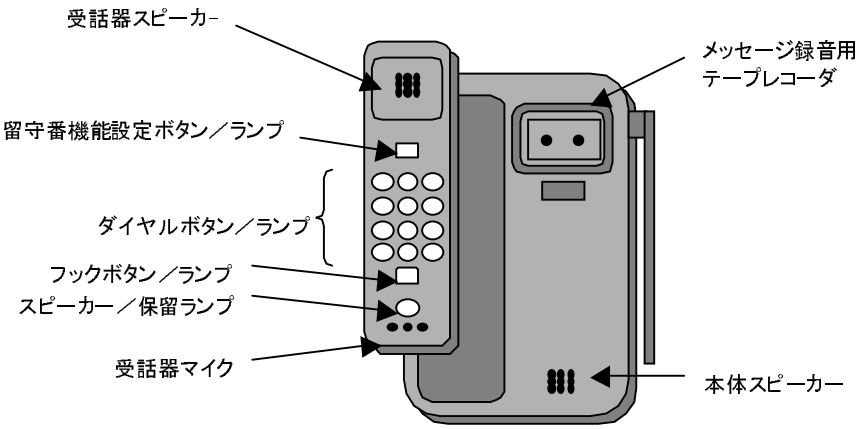


図 3.1　実装対象の電話機

図からもわかるように、受話器部分と本体部分が分離するコードレスタイプの電話機である。次に各部とはたらきを説明する。

・受話器スピーカー
　接続相手先と通話するために使用する。

・留守番機能設定ボタン／ランプ
　留守番機能をセットするときや解除するときに使用する。

・ダイヤルボタン／ランプ
　電話をかけるときに使用する。

・フックボタン／ランプ
　電話をかけるときや、相手先からの接続要求に応えるときに使用する
　フックボタンを押すことで、オフフック（相手先からの接続を受け付けない）状態や、オンフック（相手先からの接続を受け付ける）状態になる。

・スピーカー／保留ランプ
　相手先からの着信要求状態やオフフック状態を知らせるランプ

・受話器マイク
　接続相手先と通話するために使用する。

・メッセージ録音用テープレコーダー
　相手先のメッセージを録音するために使用する。

・本体スピーカー
　留守応答時の音声や、着信音などが聞こえる。

ここまでの過程は、ユーザーズマニュアルを参考にしている。しかし、このマニュアルだけでは電話機と交換機とのやり取りに関して不明な点が多い。その他の文献[5]を参考にすることで電話回線に関する調査を行った。その結果、フックボタンによって回線処理がなされていることがわかった。オフフックであるとき、回線に対して回線閉結処理を行っている。これは他からの接続要求を受け付けないようにする処理である。またオンフックであるとき、回線に対して回線解放処理を行っている。これは他からの接続要求を受け付けるようにする処理である。

３．３　電話機の仕様

対象としている電話機システムの機能は、以下の３つに分類することができる。

- 着信処理機能
- 発信機能
- 留守番機能

以下では、これら個々の機能について説明していく。

３．３．１　着信処理機能

着信処理とは、他の電話機から回線接続要求を受けた時、それに応答して電話機の着信音を鳴らす機能である。ここでは、相手先から回線接続要求を含めその後に続く、

相手先との通話、回線切断（通話終了）に至る、これら各段階の電話機で行われる処理の手順を示す。

1. 外線通話以外の状態の時、回線が外部の接続先から電話機に接続される。
   - 親機のスピーカーから着信音を発生させる。
   - スピーカー／保留ランプ（赤色）が点灯する。
   - ダイヤルボタン／ランプ（緑色）が点灯する。

   電話機利用者が受話器取って応答する以前に、相手先が接続を断った場合
   - 着信音を止める
   - スピーカー／保留ランプ（赤色）が消灯する。
   - ダイヤルボタン／ランプ（緑色）が消灯する。

2. 相手の接続要求に応えるために電話機利用者が受話器のフックボタンを押す（オフフック）と、以下の処理を行う。
   - 着信音を停止させる。
   - 受話器スピーカーがＯＮになる。
   - 受話器マイクがＯＮになる。
   - 回線閉結処理を行う。

3. 相手先と通話を行う。

4. 相手先との接続を断つために電話機利用者が受話器のフックボタンを押す。（オンフック）
   - 受話器のフックボタンを押す。
   - 受話器スピーカーがＯＦＦになる
   - 受話器マイクがＯＦＦになる
   - スピーカー／保留ランプ（赤色）が消灯する。
   - ダイヤルボタン／ランプ（緑色）が消灯する。
   - 回線開放処理を行う。

３．３．２　発信機能

発信機能とは接続したい相手の電話番号を入力し、相手先に対して接続を要求する機能である。ここでは相手先への回線接続要求を含め、その後に続く相手先との通話、回線切断（通話終了）に至るこれら各段階の電話機で行われる処理の手順を示す。

1.　電話機利用者が電話をかけるために受話器のフックボタンを押す。

（オフフック）

・受話器スピーカーをＯＮにする。
・受話器マイクをＯＮにする。
・スピーカー／保留ランプ（赤色）が点灯する。
・ダイヤルボタン／ランプ（緑色）が点灯する。
・回線閉結処理を行う。

2.　電話機利用者は、ダイヤルボタン／ランプを用いて接続したい相手先の電話番号を入力する。

・利用者によって押された数値キーを１桁ずつ発信機に送る。
・発信機が相手先電話番号をもとに、接続要求を出す。

この部分において実装では、簡易化のため以下の２つを決める。
－電話番号は１０桁と定める。
－どのような組み合わせのダイヤルボタンが押されても相手先に接続される。

また、ここでは以下の実時間制約が存在する。
―最初のダイヤルキーは３０秒以内に押されなければならない。
―２つ目以降のダイヤルキーは２０秒以内に押されなければならない。
―相手先の電話番号(１０桁)は２分以内に入力し終わらなければならない。

これらの時間制約を満たさない場合、接続要求が無効となる。
電話機利用者に無効であることを示すため、電話機では以下の処理を行う。

・ダイヤルボタン／ランプが消灯する。

接続要求が無効となった場合、フックボタンを２回押し（オンフックしてからオフフッ

ク）再度ダイヤルをやり直す必要がある。

3. 相手先が接続に答えれば、相手先と通話を行う。

4. 相手先との接続を断つために電話機利用者が受話器のフックボタンを押す。（オンフック）
   ・受話器のフックボタンを押す。
   ・受話器スピーカーがＯＦＦになる
   ・受話器マイクがＯＦＦになる
   ・スピーカー／保留ランプ（赤色）が消灯する。
   ・ダイヤルボタン／ランプ（緑色）が消灯する。
   ・回線開放処理を行う。

３．３．３　留守番機能

留守番機能とは、他の電話機から回線接続要求を受けた時、一定時間着信音を発生させた後に、自動的に回線を繋ぎ相手先メッセージの録音を開始する機能である。ここでは、相手先から回線接続要求を受け、相手先メッセージの録音、回線切断（録音終了）に至る、各段階の電話機で行われる処理の手順を示す。

1. 電話機利用者が留守番機能をセットするために留守ボタンを押す
   ・留守ボタン／ランプ（赤色）が点灯する。

2. 電話機利用者が留守番機能を解除するために留守ボタンを押す。
   ・留守ボタン／ランプ（赤色）が消灯する。

3. 外線通話以外の状態で、留守番機能がセットされているときに、回線が外部の接続先から電話機に接続される。
   ・親機のスピーカーから着信音を発生させる。
   ・スピーカー／保留ランプ（赤色）が点灯する。
   ・ダイヤルボタン／ランプ（緑色）が点灯する。
   ・回線閉結処理を行う。
   ・着信音発生から１０秒後、録音機能メッセージが流れる。
   ・レコーダーが作動し、相手先の用件が最長１分間録音される。

また、着信音発生から１０秒以内に電話機利用者がフックボタンを押した場合（オフフック）、着信機能の 3 へ移る。

4.　録音終了（回線切断）

録音が終了する条件は２つある。まず録音が１分を経過した場合であり、もう１つは、接続先の相手が１分以内に回線を断った場合である。
いずれの場合も録音終了後に以下に示す手順の処理を行う。

・メッセージ録音を終了した日付と時刻を自動的に録音する。
（タイムスタンプ機能）
・本体スピーカーをＯＦＦにする。
・スピーカー／保留ランプ（赤色）が消灯する。
・ダイヤルボタン／ランプ（緑色）が消灯する。
・回線開放処理を行う。

３．４　電話機のオブジェクト

要求仕様から抽出したオブジェクトを以下に示す。
また、オブジェクトモデルの表記法を以下の図 3.2 のように定義する。また、個々のオブジェクトを示すと共に、そのメソッドの概要についても説明する。

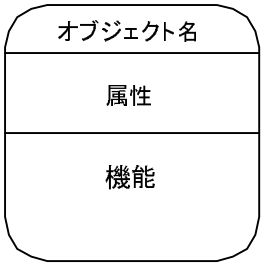
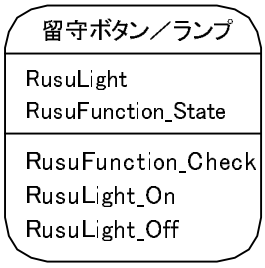


図 3.2　オブジェクトモデル表記法

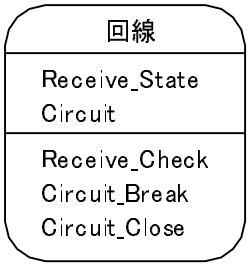
（１）留守ボタン／ランプオブジェクト



・RusuFunction_Check ： 留守番機能が押されたかどうか（セット・解除）をチェックする。
・RusuLight_On ： 留守番ボタン／ランプを点灯する。
・RusuLight_Off ： 留守番ボタン／ランプを消灯する。

以後、留守ボタン／ランプ オブジェクトを o1 で表す。

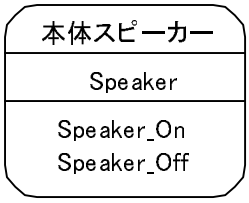
（２）回線オブジェクト



・Circuit_Break ： 回線解放処理を行う。
・Circuit_Close ： 回線閉結処理を行う。

以後、回線オブジェクトを o2 で表す。

（３）本体スピーカーオブジェクト



・Speaker_On ： 本体スピーカーを ON にする。
・Speaker_Off ： 本体スピーカーを OFF にする。

以後、本体スピーカーオブジェクトを o3 で表す。

（４）受話器スピーカーオブジェクト

・ReceiverSpeaker_On ： 受話器スピーカーを ON にする。
・ReceiverSpeaker_Off ： 受話器スピーカーを OFF にする。

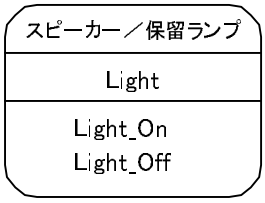
以後、受話器スピーカーオブジェクトを o4 で表す。

（５）受話器マイクオブジェクト

・ReceiverMike_On ： 受話器マイクを ON にする。
・ReceiverMike_Off ： 受話器マイクを OFF にする。

以後、受話器マイクオブジェクトを o5 で表す。

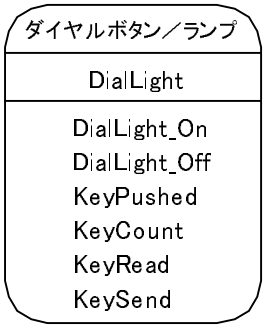
（６）スピーカー／保留ランプオブジェクト



・Light_On ： スピーカー／保留ランプを点灯させる。
・Light_Off ： スピーカー／保留ランプを消灯させる。

以後、スピーカー／保留ランプオブジェクトを o6 で表す。

（７）ダイヤルボタン／ランプオブジェクト

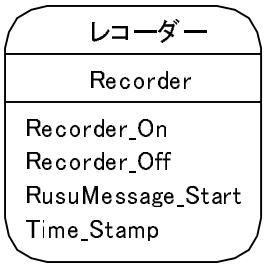


・DialLight_On ： ダイヤルボタン／ランプを点灯させる。
・DialLight_Off ： ダイヤルボタン／ランプを消灯させる。
・KeyPushed ： ダイヤルキーが押されたかどうかをチェックする。
・KeyCount ： 押されたダイヤルキーが何桁目かを識別する。
・KeyRead ： 押されたダイヤルキーを調べる。

・KeySend ： 押されたダイヤルキーを発信機に送る。

以後、ダイヤルボタン／ランプを o7 で表す。

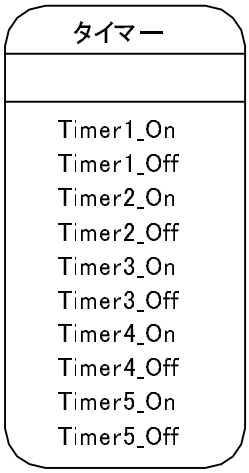
（８）レコーダーオブジェクト



・Recorder_On ： レコーダーの録音を開始させる。
・Recorder_Off ： レコーダーの録音を停止させる。
・RusuMessage_Start ： 留守番機能メッセージを発生させる。
・Time_Stamp ： メッセージ録音を行った日付と時刻を録音する。

以後、レコーダーオブジェクトを o8 で表す。

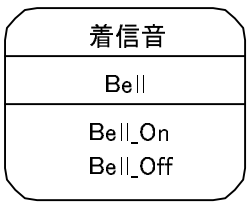
（９）タイマーオブジェクト



・Timer1_On ： 120 秒監視タイマーを ON にする。
発信処理では、相手先の電話番号を 120 秒以内に入力を終了しなければ発信処理が無効となる。この入力経過時間を監視するためのタイマー。
・Timer1_Off ： 120 監視タイマーを OFF にする。
・Timer2_On ： 30 秒監視タイマーを ON にする。
発信処理において、相手先電話番号の１桁目のダイヤルを 30 秒以内に入力しなければ発信処理は無効となる。この 30 秒を監視するためのタイマー。
・Timer2_Off ： 30 秒監視タイマーを OFF にする。
・Timer3_On ： 20 秒監視タイマーを ON にする。
発信処理において、相手先電話番号の２桁目移行のダイヤルが各々 20 秒以内に入力されなければ発信処理が無効になる。この 20 秒を監視するためのタイマー。
・Timer3_Off ： 20 秒監視タイマーを OFF にする。
・Timer4_On ： 10 秒監視タイマーON にする。
留守番機能がセットされており、着信があった場合、10 秒後にメッセージ録音を開始するための時間監視タイマー。
・Timer4_Off ： 10 秒監視タイマーを OFF にする。
・Timer5_On ： 60 秒監視タイマーを ON にする。
メッセージ録音時間（最長１分間）を監視するためのタイマー。
・Timer5_Off ： 60 秒監視タイマーを OFF にする。

以後、タイマーオブジェクトを o9 で表す。

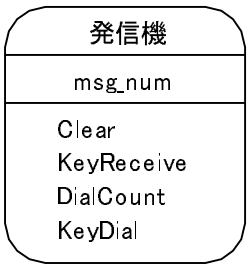
（１０）着信音オブジェクト



・Bell_On ： 着信音を発生させる。
・Bell_Off ： 着信音を停止させる。

以後、着信音オブジェクトを o10 で表す。

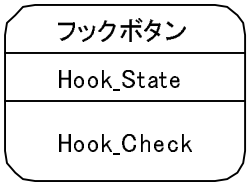
（１１）発信機オブジェクト



・Clear ： ダイヤルキーを記憶しているメモリをクリアする。
・KeyReceive ： 押されたダイヤルキーを受け取る。
・DialCount ： 押されたダイヤルキーの数をカウントする。
・KeyDial ： 電話番号を発信する。

以後、発信機オブジェクトを o11 で表す。

（１２）フックボタンオブジェクト



・Hook_Check ： フックボタンの状態（オンフック・オフフック）をチェックする。

以後、フックボタンオブジェクトを o12 で表す。

３．５　電話機の SES モデル

SES を基にしてシステムの振る舞いを定義するモデルが SES モデルである。状態遷移図の各状態に SES を割り当てることで SES モデルを作成する。SES モデルを作成するために、電話機システムにおける SES の抽出を行う。

３．５．１　電話機の SES

ここまでの作業から以下に示す図 3.3 のような状態遷移図が得られた。

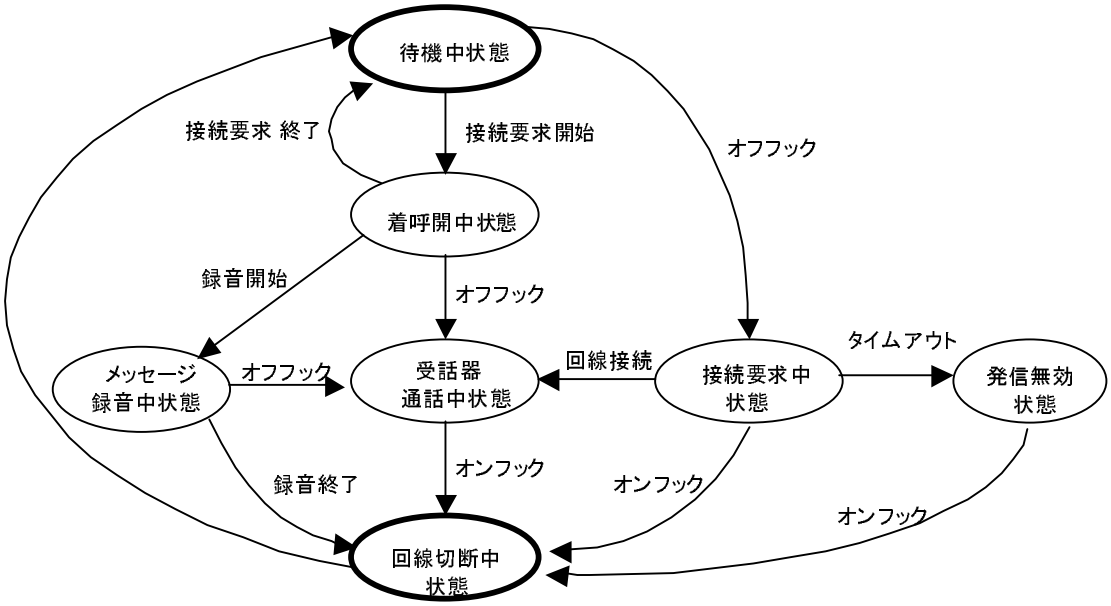


図 3.3　電話機における状態遷移図

この状態遷移図を基に SES の抽出を行う。これより以下に抽出した SES を示す。実際に得られた SES は１５個あり、各 SES を識別するために名前を付けている。さらに、SES の処理の流れを図で示し、処理列の１つ１つに対してオブジェクトの機能を割り当てている。また、状態遷移図に示されている状態間を結ぶ矢印はイベントであり、矢印の横にイベント名を示している。

（１）SES-RUSU_FUNCTION

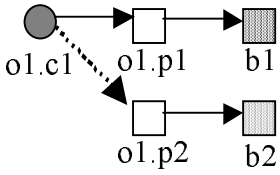


図 3.4　SES-RUSU_FUNCTION の処理の流れ

・o1.c1 ： RusuFunction_Check
・o1.p1 ： RusuLight_On
・b1 ： 先頭の処理に戻る
・o1.p2 ： RusuLight_Off
・b2 ： 先頭の処理に戻る

（２）SES-RECEIVE1

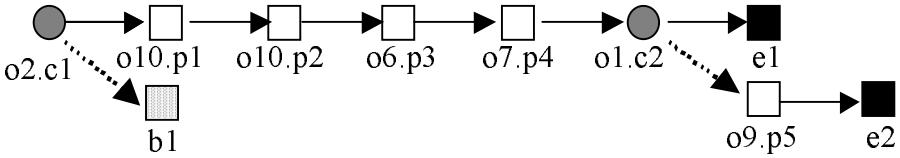


図 3.5　SES-RECEIVE1 の処理の流れ

・o2.c1 ： Receive_Check
・o10.p1 ： Speaker_On
・o3.p2 ： Bell_On
・o6.p3 ： Light_On
・o7.p4 ： DialLight_On
・c2 ： RusuFunction_Check
・o9.p5 ： Timer4_On
・e1 ： イベントを出力する
・e2 ： イベントを出力する
・b3 ： 先頭の処理に戻る

（３）SES-RECEIVE3

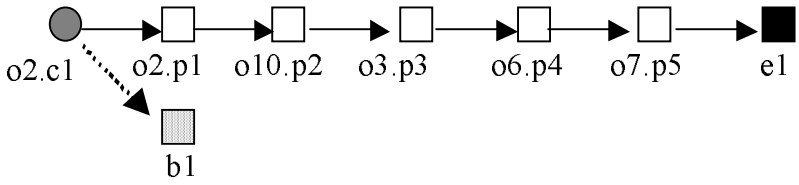


図 3.6　SES-RECEIVE3 の処理の流れ

・o2.c1 ： Receive_Check
・o2.p1 ： Timer4_Off
・o10.p2 ： Bell_Off
・o3.p3 ： Speaker_Off
・o6.p4 ： Light_Off
・o7.p5 ： DialLight_Off
・e1 ： イベントを出力する
・b2 ： 先頭の処理に戻る

（４）SES-RECEIVE3

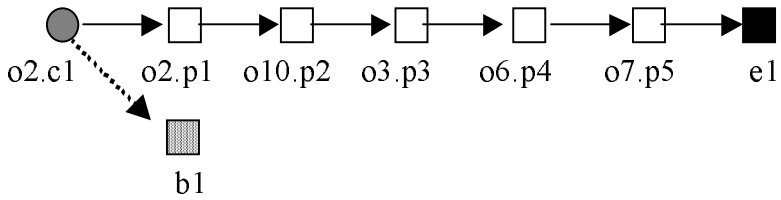


図 3.7　SES-RECEIVE3 の処理の流れ

・o2.c1 ： Receive_Check
・o9.p1 ： Timer5_Off
・o4.p2c ： Recorder_Off
・o3.p3 ： Speaker_Off
・o6.p4 ： Light_Off
・o2.p5 ： Circuit_Break

・e1 ： イベントを出力する
・b2 ： 先頭の処理に戻る

（５）SES-HOOK1

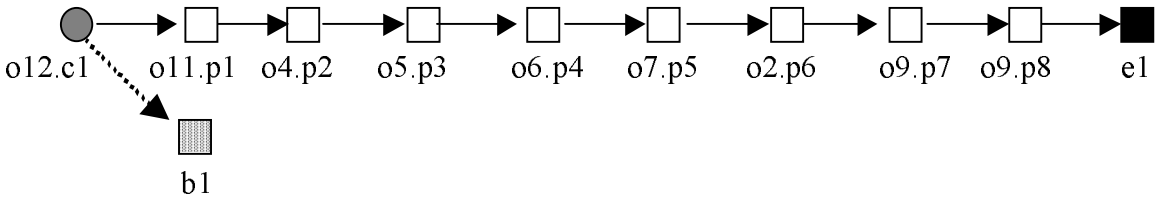


図 3.8　SES-HOOK1 の処理の流れ

・o12.c1 ： Hook_Check
・o11.p1 ： Clear
・o4.p2 ： ReceiverSpeaker_On
・o5:p3 ： ReceiverMike_On
・o6.p4 ： Light_On
・o7.p5 ： DialLight_On
・o2.p6 ： Circuit_Close
・o9.p7 ： Timer1_On
・o9.p8 ： Timer2_On
・e1 ： イベントを出力する
・b2 ： 先頭の処理に戻る

（６）SES-HOOK2

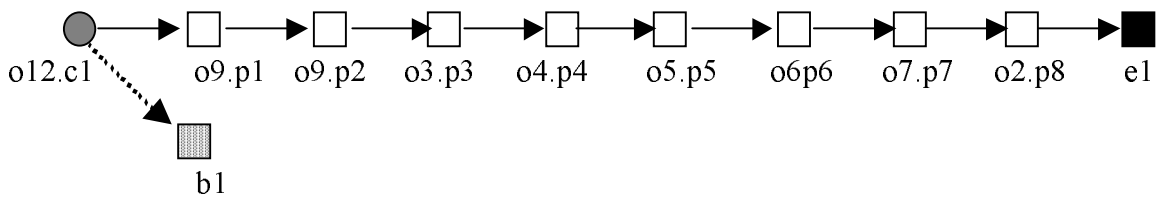


図 3.9　SES-HOOK2 の流れ

・o12.c1 ： Hook_Check
・o9.p1 ： Timer1_Off
・o9.p2 ： Timer2_Off
・o9.p3 ： Timer3_Off
・o4.p4 ： ReceiverSpeaker_Off
・o5.p5 ： ReceiverMike_Off
・o6.p6 ： Light_Off
・o7.p7 ： DialLight_Off
・o2.p8 ： Circuit_Break
・e1 ： イベントを出力する
・b2 ： 先頭の処理に戻る

（７）SES-HOOK3

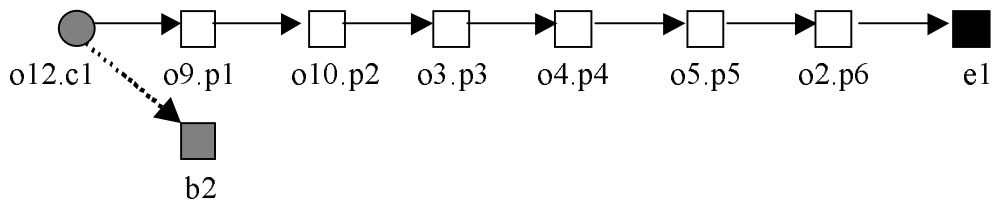


図 3.10　SES-HOOK3 の処理の流れ

・o12.c1 ： Hook_Check
・o9.p1 ： Timer4_Off
・o10.p2 ： Bell_Off
・o3.p3 ： Speaker_Off
・o4.p4 ： ReceiverSpeaker_On
・o5.p5 ： ReceiverMike_On
・o2.p6 ： Circuite_Close
・e1 ： イベントを出力する
・b2 ： 先頭の処理に戻る

（８）SES-HOOK4

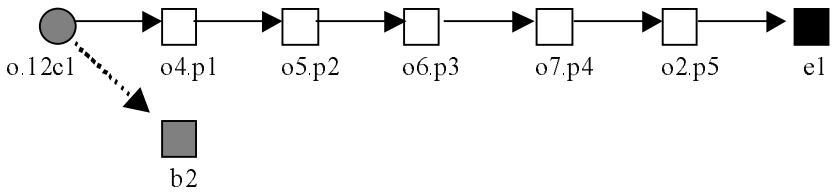


図 3.11　SES-HOOK4 の処理の流れ

・o12.c1 ： Hook_Check
・o4.p1 ： ReceiverSpeaker_Off
・o5.p2 ： ReceiverMike_Off
・o6.p3 ： Light_Off
・o7.p4 ： DialLight_Off
・o2.p5 ： Circuit_Close
・e1 ： イベントを出力する
・b1 ： 先頭の処理に戻る

（９）SES-HOOK5

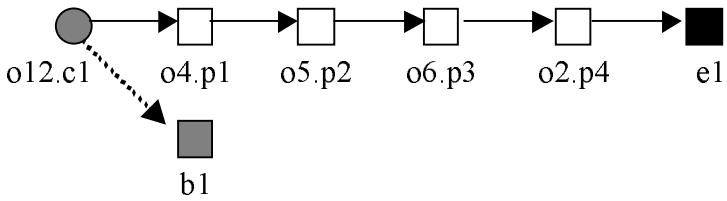


図 3.12　SES-HOOK5 の処理の流れ

・o12.c1 ： Hook_Check
・o4.p1 ： ReceiverSpeaker_Off
・o5.p2 ： ReceiverMike_Off
・o6.p3 ： Light_Off
・o2.p4 ： Circuit_Close
・e1 ： イベントを出力する

・b1 ： 先頭の処理に戻る

（１０）SES-HOOK6

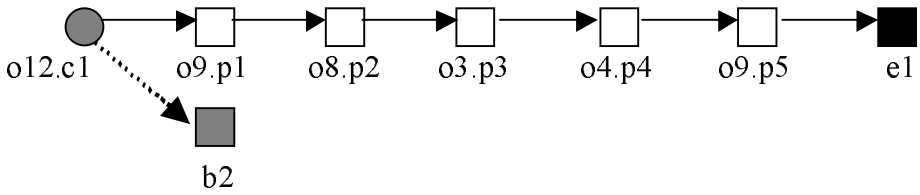


図 3.13　SES-HOOK6 の処理の流れ

・o12.c1 ： Hook_Check
・o9.p1 ： Timer5_Off
・o8.p2 ： Recorder_Off
・o3.p3 ： Speaker_Off
・o4.p4 ： ReceiverSpeaker_On
・o5.p5 ： ReceiverSpeaker_On
・e1 ： イベントを出力する
・b1 ： 先頭の処理に戻る

（１１）SES-DIAL_NUMBER

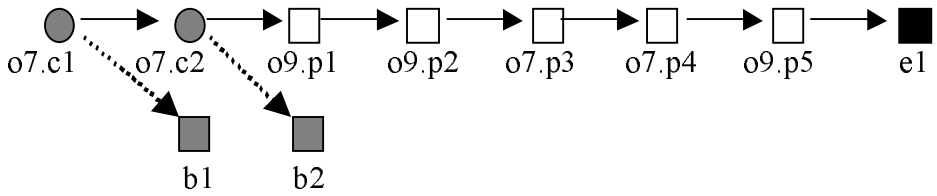


図 3.14　SES-DIAL_NUMBER の処理の流れ

・o7.c1 ： KeyPushed
・o7.c2 ： KeyCount
・o9.p1 ：Timer2_Off

・o9.p2 ： Timer3_On
・o7.p3 ： KeyRead
・o7.p4 ： KeySend
・o9.p5 ： Timer2_On
・e1 ： イベントを出力する
・b1 ： 先頭の処理に戻る
・b2 ： 先頭の処理に戻る

（１２）SES-DIAL

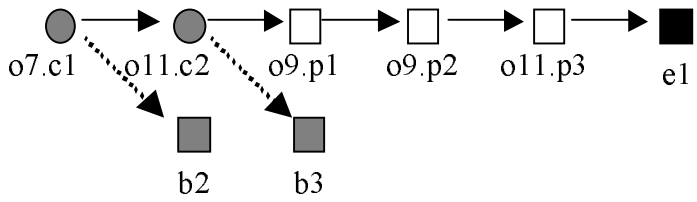


図 3.15　SES-DIAL の処理の流れ

・o7.c1 ： KeyReceive
・o11.c2 ： DialCount
・o9.p1 ： Timer1_Off
・o9.p2 ： Timer3_Off
・o11.p3 ： KeyDial
・e1 ： イベントを出力する
・b2 ： 先頭の処理に戻る
・b3 ： 先頭の処理に戻る

（１３）SES-TIMEOUT

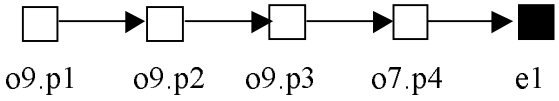


図 3.16　SES-TIMEOUT の処理の流れ

・o9.p1 ： Timer1_Off
・o9.p2 ： Timer2_Off
・o9.p3 ： Timer3_Off
・o7.p4 ： DialLight_Off
・e1 ： イベントを出力する

（１４）SES-RECORDER_ON

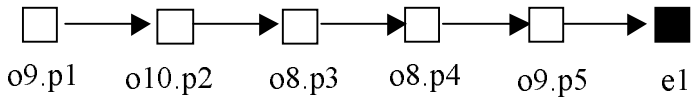


図 3.17　SES-RECORDER_ON の処理の流れ

・o9.p1 ： Timer4_Off
・o10.p2 ： Bell_Off
・o8.p3 ： RusuMessage_Start
・o8.p4 ： Recorder_On
・o9.p5 ： Timer5_On
・e1 ： イベントを出力する

（１５）SES-RECORDER_OFF

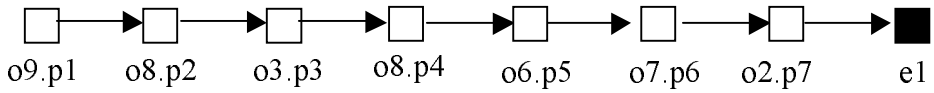


図 3.18　SES-RECORDER_OFF の処理の流れ

・o9.p1 ： Timer5_Off
・o8.p2 ： Recorder_Off
・o3.p3 ： Speaker_Off
・o8.p4 ： Time_Stamp
・o6.p5 ： Light_Off

・o7.p6 ： DialLight_Off
・o2.p7 ： Circuit_Break
・e1 ： イベントを出力する

３．５．２　SES モデルの作成

上の節で定義した SES はシステム全体の振る舞いの断片であり、これら SES の集合を適切に並べることによりシステムの振る舞いは定義される。よって、状態遷移図の各状態に適切な SES を割り当て、SES モデルを作成する。図 3.19 に作成した SES モデルを示す。定義した SES の処理列全てを状態遷移図に描くのは困難であるため、SES 名のみを各状態に割り当てている。

図に示した状態遷移図からもわかるように以下に示す３つは実時間制約によるイベントにより状態が遷移する。

・接続要求中状態から発信無効状態
　接続要求中状態で２０秒（または、３０秒、１２０秒）経過後に、発信無効状態に遷移する。

・呼着開中状態からメッセージ録音中状態
　呼着開中状態から１０秒経過後に、メッセージ録音中状態に遷移する。

・メッセージ録音中状態から回線切断状態
　メッセージ録音中状態から６０秒経過後に、回線切断状態に遷移する。

このように、ある時間が経過した後に状態遷移が起きるようなしくみを、実装では周期起動ハンドラにより実現している。この詳細については第４章で述べる。

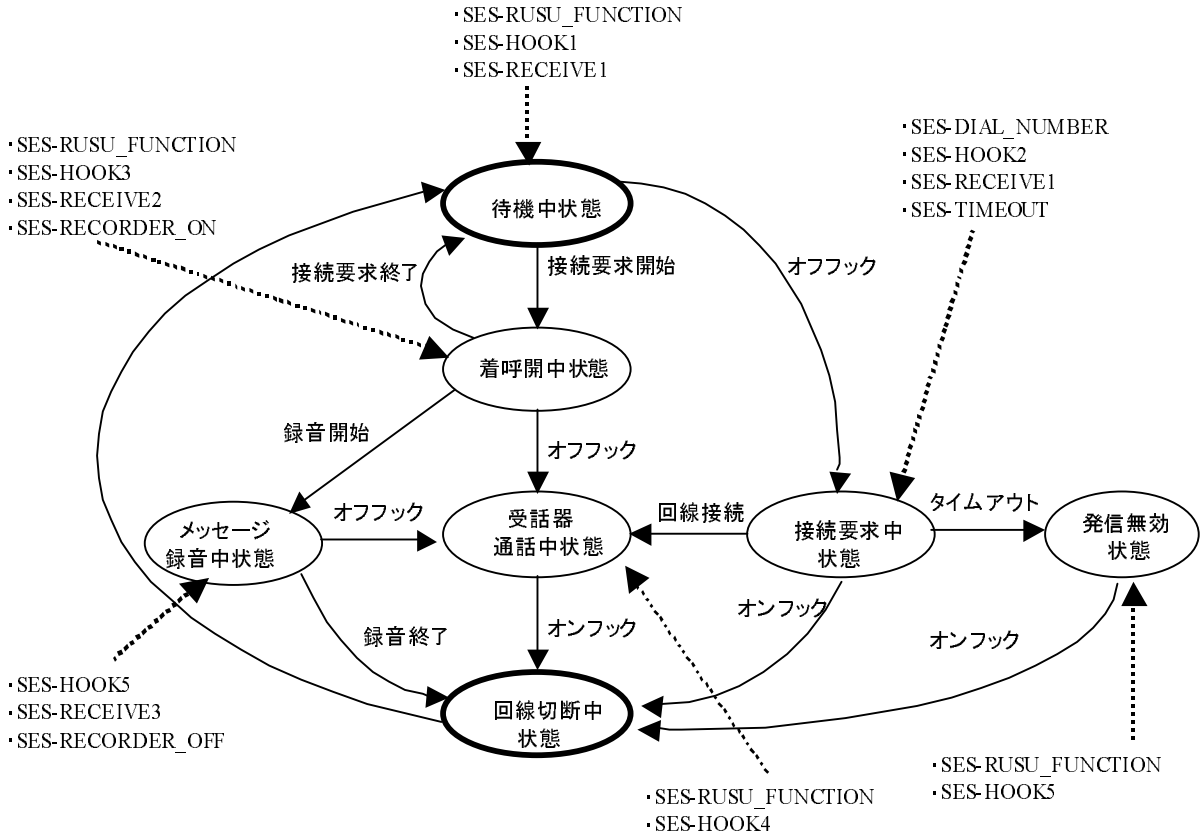

・SES-RUSU_FUNCTION
・SES-HOOK1
・SES-RECEIVE1
・SES-RUSU_FUNCTION
・SES-HOOK3
・SES-RECEIVE2
・SES-RECORDER_ON
・SES-DIAL_NUMBER
・SES-HOOK2
・SES-RECEIVE1
・SES-TIMEOUT
待機中状態
接続要求終了
接続要求開始
オフフック
着呼開中状態
録音開始
オフフック
メッセージ
録音中状態
オフフック
受話器
通話中状態
回線接続
接続要求中
状態
タイムアウト
発信無効
状態
録音終了
オンフック
オンフック
オンフック
・SES-HOOK5
・SES-RECEIVE3
・SES-RECORDER_OFF
回線切断中
状態
・SES-RUSU_FUNCTION
・SES-HOOK4
・SES-RUSU_FUNCTION
・SES-HOOK5

図 3.19　SES モデル

第４章

電話機システムの実装法

4.1　実装環境

ITRON(Industrial-The Real-time Operating system Nucleus)[6]は機器組み込み制御システム用のオペレーティングシステムであり、この業界におけるデファクト・スタンダードの地位を確立している。そこで本研究では実装環境として ItIs(ITRON Implementation by Sakamura Lab)を用いている。ItIs とは東京大学 坂村研究室で研究・教育用に開発され、豊橋技術大学 組み込みリアルタイムシステム研究室で開発が続けられており、ITRON 仕様の最新版であるμITRON3.0 仕様[7]に準拠したリアルタイムカーネルである。また、実際の組み込みシステム開発ではアセンブリ言語と C 言語による実装が一般的に行われているが、本研究では実装言語としてＣ言語のみを用いている。

4.2　ItIs

ここでは、本研究の実装で使用している ItIs について説明する。

4.2.1　タスク管理機能

ITRON 仕様では、並行処理されるプログラムの単位をタスクと呼び、タスクを管理するための主な情報には、タスク ID（タスクの名前）、タスクの状態、タスクの優先度などがある。また、タスクの状態はタスクの状態遷移に関係し、タスクの優先度はタスクのスケジューリングに関係している。

4.2.2　タスクの状態

タスクの取り得る状態として、実行状態(RUN)、実行可能状態(READY)、待ち状態(WAIT)の３つの基本状態に加えて、強制待ち状態(SUSPEND)、二重待ち状態(WAIT-SUSPEND)、休止状態(DORMANT)、未登録状態(NON_EXISTENT)の７つの状態を定義している。タスクの状態を以下に図 4.1 で示す。

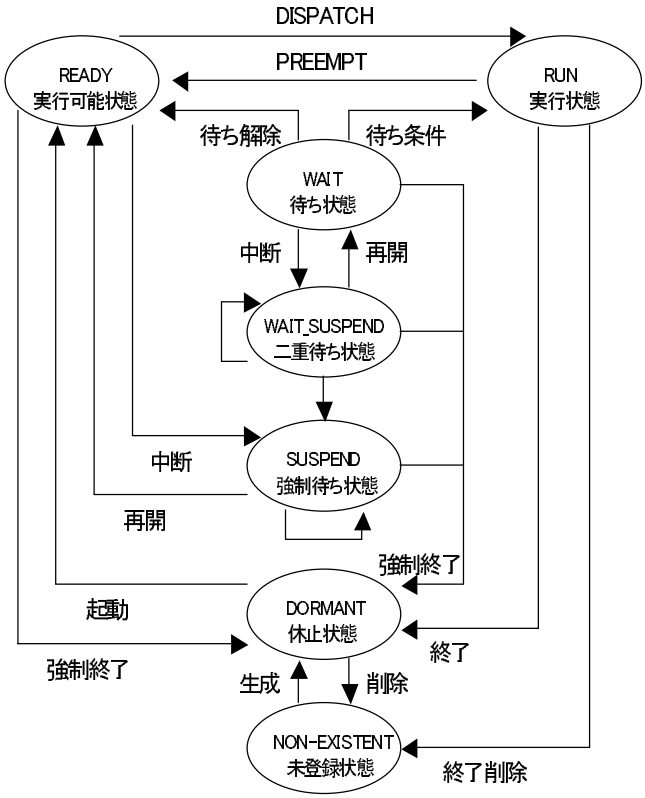


図 4.1　タスク状態遷移図

4.2.3　スケジューリング

タスクのスケジューリングは、タスクの優先度を基準にして行われ、実行可能状態(READY)にあるタスクの中で最も高い優先度を持つタスクが実行状態に(RUN)となる。実行可能状態(READY)および実行状態(RUN)のタスクは、タスクの優先度順に行列を作っており、その行列（レディーキューと呼ぶ）の先頭タスクが実行状態(RUN)となる。この様子を図 4.2 に示す。

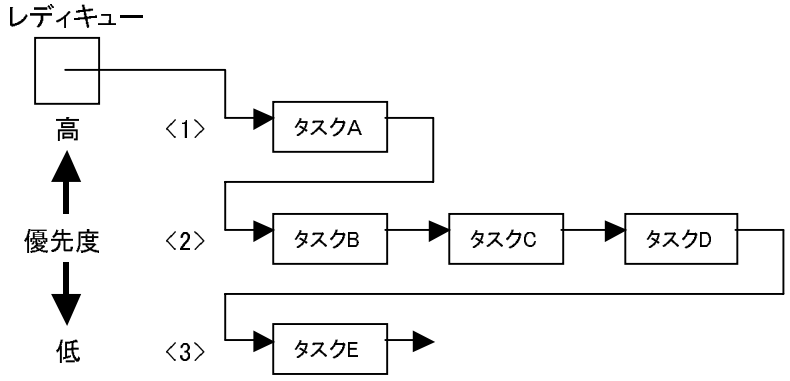


図 4.2　レディキュー

ITRON 仕様 OS では優先度ベースのスケジューリングを採用しているため、この優先度は絶対的な意味をもつ。いかなる場合でもレディキューの先頭のタスクしか実行されない。すなわち、高い優先度を持つタスクが実行されている間は、それより低いタスク優先度を持つタスクはまったく実行されない。逆に、低い優先度のタスクが実行されている間に高い優先度のタスクが実行可能状態になると、実行中だった低優先度のタスクは中断され（実行状態から実行可能状態に移る）、高い優先度のタスクを先に実行する。また、同じ優先度のタスクが複数存在した場合には、First Come First Served 方式（最初に来たものから処理される）原則によるスケジューリングが行われる。すなわち、先に実行可能状態になったタスクが先に実行状態になる。この場合でも、レディキューの先頭にないタスクは、それがレディキューの先頭のタスクと同じ優先度であったとしても、まったく実行されない。レディキューの２番目以下のタスクが実行されうには、レディキューの先頭のタスクがレディキューから抜ける（待ち状態など他のタスク状態に移る）必要がある。

4.3　SES モデルの実装法

4.3.1　SES モデルをタスクとして実装するためのアーキテクチャ

SES アプローチでは SES を処理の単位としており、RTOS ではタスクを処理の単位としている。ここでは、SES をタスクとして実装する手法を説明する。SES アプローチにおけるタスクのアーキテクチャと SES モデルがどのように対応しているのかを図 4.3 に示す。

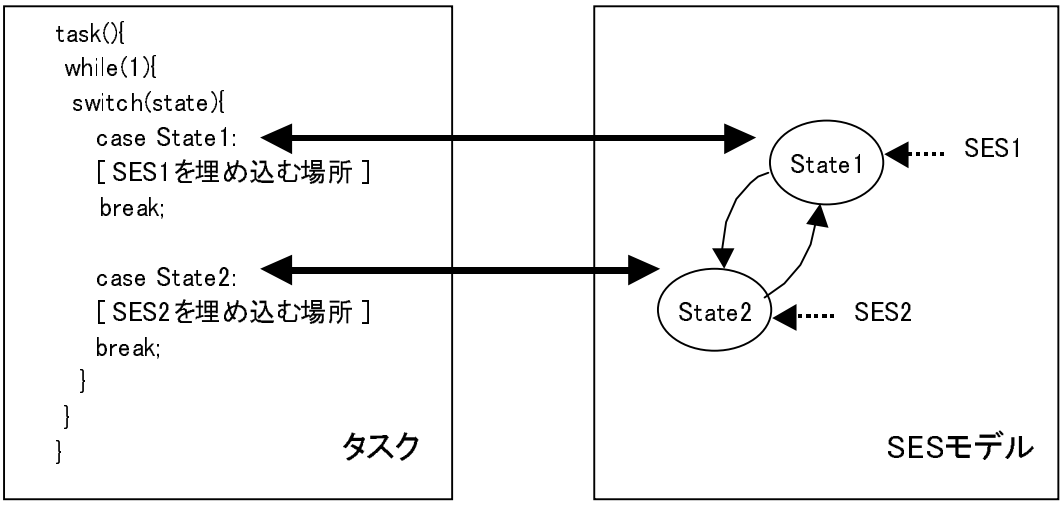


図 4.3　タスクと SES モデルの対応

タスクは無限ループのブロックを持ち、この無限ループの内部には switch ブロックが存在する。SES 設計モデルの各状態は、swtich ブロックにおける case により実現する。図の例では SES モデルは２つの状態（State1、State2）をもつため、タスクでは２つの case 文(case1、case2)により実現される。また、各 case に SES コンポーネントが埋め込まれる。この図では、SES モデルの State1 に SES1 が存在するため、タスクの State1 の case ブロックに SES1 が埋め込まれる。また、同様に SES モデルの SES2 はタスクの State2 の case ブロックに SES2 が埋め込まれる。
次に、状態遷移がどのように行われるのかについて説明する。状態遷移の実現は、大域変数 int state によって行っている。状態遷移図の各状態は整数で表しているため、状態を遷移させるためには現在の state 変数の値とは異なる値を代入することで実現する。

4.3.2　SES の実装法

ここまでは、SES モデルを RTOS 上で実現するためのアーキテクチャを述べた。次に、SES モデルに付随する SES の実装法について説明する。ここでは、第３章で抽出された SES がどのように RTOS 上で実現されるかについて、SES-HOOK5 を例として説明する。

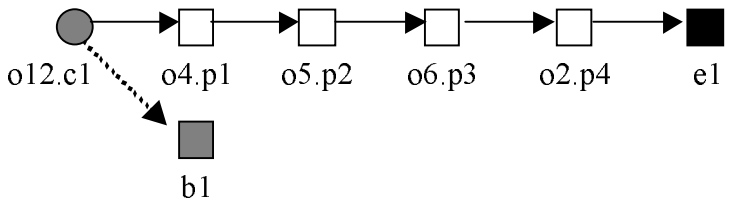


図 4.4　SES-HOOK5 の処理の流れ

図 4.4 で示しているオブジェクトの識別子は、以下のものに対応している。

・o12：フックボタン／ランプ
・o4:受話器スピーカー
・o5:受話器マイク
・o6:スピーカー／保留ランプ
・o2:回線

次に SES-HOOK5 が示す各処理の説明と、RTOS 上に実装した各処理に該当する関数名を以下に挙げる。

・o12.c1 ： フックボタンの状態に関する条件分岐　[Hook_Check()]
　　オンフックならデフォルトの処理
・o4.p1 ： 受話器スピーカーをＯＦＦにする　[ReceiverSpeaker_Off()]
・o5.p2 ： 受話器マイクをＯＦＦにする　[ReceiverMike_Off()]
・o6.p3 ： スピーカー／保留ランプをＯＦＦにする　[Light_Off()]
・o2.p4 ： 回線解放処理を行う　[Circuit_Break()]

図 4.4 に示す SES-HOOK5 を実装したものが、以下に示す図 4.5 である。SES-HOOK5 の各処理は、RTOS 上のタスクにおいて関数呼び出しという形で実装している。図 4.5

のコードに(1)～(7)の番号を割り当て、この番号順に従って、SES-HOOK5 の処理列を以下に説明する。

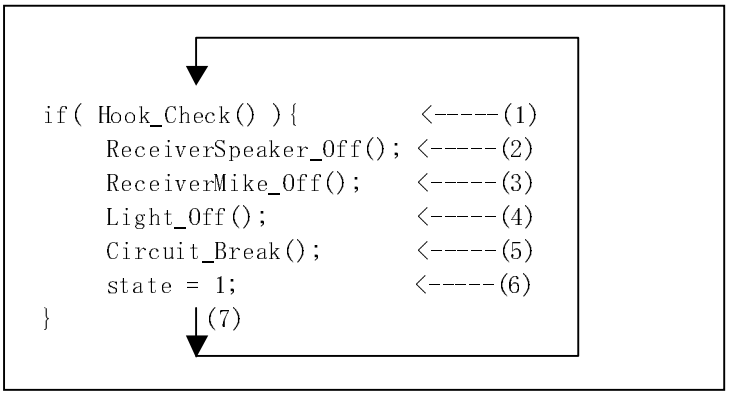


図 4.5　SES-HOOK5 の実装例

（1）　o12.c1

これは条件分岐処理である。そのため if 文により実現される。ここではフックの状態に依存して条件分岐処理が行われる。フックの状態がオンフックされていればデフォルトの処理である（2）～（6）が行われる。オフフック状態のままだと例外処理である（7）が行われる。

Hook_Check()は int 型の値を返す関数として表される。この返り値によってデフォルトと例外の処理が区別される。以下に実装したコードの概要を示す。

フック状態は整数型の大域変数 Hook_State により表現されている。変数が 1 の時、オンフックであり、0 のときオフフックの状態を意味する。この Hook_Check メソッドでは、Hook_State の変数が 1 のときにフックをオンフック状態にしている。

```
int Hook_Check(){
      if(Hook_State == 1){
            Hook_State = 0;
            return 1; （オンフックなら1を返す。）
}else
            return 0; （オフフックなら0を返す。）
      }
}
```

（２）　o4.p1

受話器スピーカーを OFF にする処理を表している。これを実現するコードの概要を以下に示す。電話機はコンピュータ上で仮想的に実現されているため、printf 文によって動作の振る舞いを表現している。以下に続く(3)から(5)でも printf 文を用いているが、これと同じ理由による。受話器スピーカーは整数型の大域変数 ReceiverMike により表現してている。変数が１の時、受話器スピーカーが ON であり、０のとき OFF にされているものとする。この ReceiverSpeaker_Off メソッドでは、ReceiverSpeaker の変数が０のとき受話器スピーカーを OFF にしている。

```
void ReceiverSpeaker_Off(){
     if(ReceiverSpeaker == 0){
          printf(“Speaker Off ¥n¥r”);
          ReceiverMiek = 1;
     }
}
```

（３）　o5.p2

受話器マイクを OFF にする処理を表している。これを実現するコードの概要を以下に示す。受話器マイクは整数型の大域変数 ReceiverMike により表現されている。変数が１の時、受話器マイクが ON であり、０のとき OFF にされているものとする。この ReceiverMike_Off メソッドでは、ReceiverMike の変数が０のとき受話器マイクを OFF にしている。

```
void ReceiverMike_Off(){
    if(ReceiverMike == 0){
         printf(“Mike Off ¥n¥r”);
         ReceiverMike = 1;
    }
}
```

（４）　o6.p3

スピーカー／保留ランプを OFF にする処理を表している。これを実現するコードの概要を以下に示す。スピーカー／保留ランプは整数型の大域変数 Light により表現されている。変数が１の時、ランプが点灯しており、０のとき消えているものとする。この Light_Off メソッドでは、Light の変数が０のとき、スピーカー保留ランプを消灯させる動作を行っている。

```
void Light_Off(){
    if(Light == 0){
        printf("Light Off ¥n¥r");
       Light = 1;
    }
}
```

（５）　o2.p4

回線を解放する処理を表している。これを実現するコードの概要を以下に示す。回線は整数型の大域変数 Circuit により表現されている。変数が１の時、回線が解放状態であり、０のとき回線が閉結状態であるものとする。この Circuit_Break メソッドでは、Circuit の変数が 1 のとき回線を解放する処理を行っている。

```
void Circuit_Break(){
    if(Circuit == 1){
        printf("Break the Circuit ¥n¥r");
        Circuit = 0;
    }
}
```

（6） e1

イベントを出力する処理である。SES モデルはイベントにより他の状態に遷移する。この状態遷移を実現するために、先に説明した整数型の大域変数 state の値を変更する。

（7） b1

先頭の処理に戻ることを意味している。よって、先頭の処理である o12.c1 の条件分岐処理に戻る。

4.3.3 外部環境とのインターフェースを実現するタスク

実装では、例外的に外部環境とのインターフェースを実現するタスクを設けている。以下に示す 1 から 7 の振る舞いを独立したタスクとしてエミュレートするようにしている。

1. 相手先からの接続要求開始
2. 相手先からの接続要求終了
3. オンフック
4. オフフック
5. 留守番機能セット
6. 留守番機能解除
7. 電話番号入力

実装しているコードレスホンシステムは、仮想的にその動作をコンピュータ上でシミュレーションさせている。そのため、実装ではキーボードからのキー入力を受け付けることにより、システム作動のトリガーとしている。

例えばフックボタンをキーボードの TAB キーに割り当て、このキーが押されることでオフフック、オンフックを実現している。よって、フックボタン、留守番機能ボタン、電話番号（ダイヤルボタン）、相手先からの接続要求開始・終了を意味するキーをそれぞれ割り当てることでシステムの様々な振る舞いを実現している。

キーボードからのキー入力を受け付ける処理は、システムコール(serial_read)によって実現している。同一状態でこれらのシステムコールや関数を使用するタスクが複数あった場合、キーボード入力に対するブロックが起こるため正常なシステムの振

る舞いを実現することが不可能となる。これを回避するために、外部環境の事象（上の１から７で示したもの）を集中的に捉えるタスクを特別に設けている。この TASK に埋め込まれている処理列は、上で示した 1 から 7 に各々割り当てているキーを認識するものであり、タスク内部の各 case ブロックにキーボードからのキーを受け付ける処理を設けている。今回の実装では、Task5 は外部環境とのインターフェースを実現するタスクとして位置付けている。

ここで、実際に上に挙げた１から７が、キーボードのどのキーに対応しているのか、また、そのキーが押されることでどのように振る舞うのかを以下で説明する。

（1）

1. 相手先からの接続要求開始
2. 相手先からの接続要求終了

この２つを仮想的に実現するため、キーボードの“：”のキーにこれらを対応させている。接続要求がない状態で、“：”キーを押すと、接続要求開始となる。また、接続要求開始の状態で再びこのキーが押されると、接続要求終了となる。

（2）

3. オンフック
4. オフフック

この２つを仮想的に実現するために、キーボードの“／”のキーにこれらを対応させている。オンフックの状態で、“／”キーが押されるとオフフック状態になり、逆に、オフフックの状態で“／”キーが押されるとオンフック状態になる。

（3）

5. 留守番機能セット
6. 留守番機能解除

この２つを仮想的に実現するために、キーボードの“；”キーにこれらを対

応させている。留守番機能がセットされていない状態で“;”キーが押されると、留守番機能がセットされ、逆に留守番機能がセットされた状態で“;”キーが押されると、留守番機能が解除されることを意味する。

（4）

7. 電話番号入力

電話番号を入力するためには、電話機のダイヤルキーを押すことになるが、これを仮想的に実現するために、キーボードの“０”、“１”、“２”、“３”、“４”、“５”、“６”、“７”、“８”、“９”に電話機のダイヤルキーとして対応させている。これらのキーを押すことで電話番号を仮想的に入力する。

4.3.4 ItIs のシステムコール

実装で使用した ItIs のシステムコールを以下に示すとともに、その概要を述べる。

・wup_tsk：他タスクの起床。

・slp_tsk：自タスクを起床待ち状態へ移行する。

・tslp_tsk：自タスクを起床待ち状態へ移行（タイムアウト有り）する。

・cre_mbf：メッセージバッファを生成する。

・del_mbf：メッセージバッファを削除する。

・snd_mbf：メッセージバッファへメッセージを送信する。

・rcv_mbf：メッセージバッファからメッセージを受信する。

・cre_tsk：タスクを生成する。

・sta_tsk：タスクを起動する。

・ter_tsk：タスクを強制終了する。

・del_tsk：タスクを削除する。

・chg_pri：タスクの優先度を変更する。

・def_cyc：周期起動ハンドラを定義する。

・act_cyc：周期起動ハンドラの活性状態を制御する。

4.4　ItIs 上での SES モデルの実装法

先に SES をタスクとして実現する方法を説明した。図 4.6 に示すように、各状態に付随する SES はタスクに詰め込む必要があるが、これらの SES をどのタスクに詰め込むかを決定しなければならない。SES のタスクへの詰め込み方によって、スケジューリングやタスク間通信の実現方法に影響を及ぼす。例えば、スケジューリングに関して、実時間制約をもつ SES が複数存在するとき、このような SES の制約を満たすために、時間制約のある SES を同一のタスクに詰め込む場合と、別々のタスクに詰め込む場合とではスケジューリングの方法が異なる。また、タスク間通信では、以下の図に示しているように、SES3 と SES8 が通信（データのやり取り）を行うような場合、この SES3 と SES8 が同一のタスクに詰め込まれれば、タスク内の局所変数により通信を行うことが可能となる。また逆に、別々のタスクに詰め込まれれば、メッセージバッファやメイルボックスによる通信方法を行う必要がある。

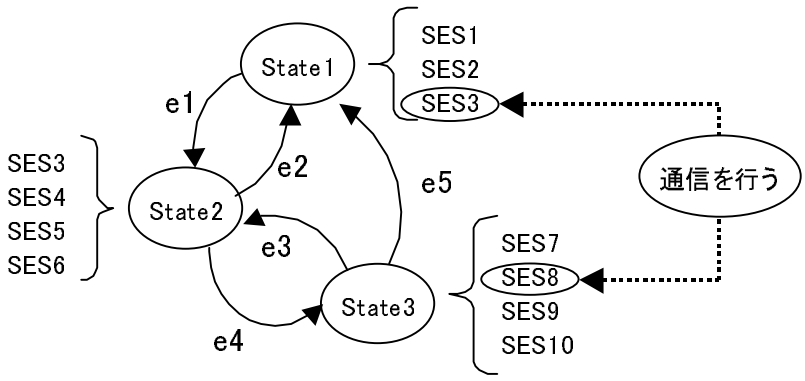


図 4.6　SES モデル

よって、SES モデルを RTOS 上で実装するために以下３つを考慮する必要がある。

- SES をどのようにタスクへ詰め込むか。
- SES の実行をどのように行うか。
- SES 間の通信をどのように行うか。

そこで、本研究ではアスペクト指向プログラミングによってこれらの要件を扱う。

4.5　アスペクト指向プログラミング

実装ではアスペクト指向プログラミング（AOP ： Aspect-Oriented Programming）に着目し、設計モデルから最終的なソフトウェアを獲得するまでの流れを説明する。AOP のアプローチでは、コンポーネントプログラムでは記述しづらい側面をアスペクトプログラムとして記述し、コンポーネントプログラムとアスペクトプログラムを Weave することによって最終的なソフトウェアを得る。

本研究では、SES のタスクへの詰め込み方、タスクのスケジューリング、タスク間通信という３点のアスペクトに着目し、以下に示すように、これらのアスペクト記述言語を決定する。

・ SES をどのようにタスクへ詰め込むか
　　タスクへの詰め込み規則記述言語を決定する。

・ SES の実行をどのように行うか
　　スケジューリングアスペクト記述言語を決定する。

・ SES・間の通信をどのように行うか
　　タスク間通信のアスペクト記述言語を決定する。

また、本アプローチでは以下に示す２種類のコンポーネントに分類して考える。

・機能コンポーネント
　Ｃ言語の関数と変数の集合。

・SES コンポーネント
　SES で表現する処理列の実装であり、処理列はオブジェクトの持つメソッドにより構成されている。

AOP では、コンポーネントプログラムとアスペクトプログラムを Weaver と呼ばれる合成器にかけ、最終的なプログラムを自動的に生成するものであるが、本研究では、実行可能なプログラムを得るために手動による Weaver でこれを実現している。このことを踏まえた上で、Weave 後に最終的なソフトウェアが獲得できる実装法の流れを示す。

4.6　タスクへの詰込み

4.6.1　タスクへの詰込み規則記述言語

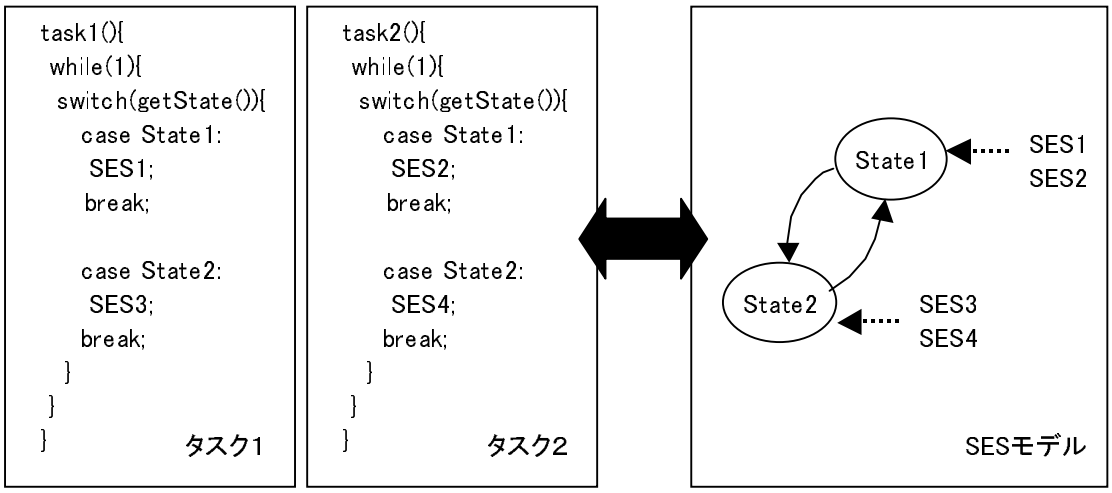


図 4.7　タスクと SES モデルの対応

図 4.7 の State1 では２つの SES(SES1、SES2)が付随している。よって、これをタスクとして RTOS 上で実装するには２つのタスク(Task1、Task2)が必要となる。この図の例からもわかるように、状態に付随する SES の数だけタスクが必要となる。また、この例では Task1 に SES1、SES3 を詰め込み、Task2 に SES2、SES4 を詰め込んでいる。しかし、Task1 に SES2、SES3、Task2 に SES1、SES4 として詰め込むことも可能である。よって、状態に付随する SES をどのタスクに詰め込むのかを決定めなければならない。SES コンポーネントのタスクへの詰め込みを決めるため、以下に示すタスクへの詰め込み規則記述言語を定義する。

・タスクへの詰込み規則記述言語

[SES_ID] ⇒　TaskID

ここで、SES_ID は SES コンポーネントの SES 名であり、この[]内に SES 名を記述する。さらに[]内に記述した SES コンポーネントがどのタスクに埋め込まれるのかを TaskID により決定する。

4.6.2　タスクへの詰め込み規則記述言語の Weave

先に定義した詰込み規則記述言語によりSESモデルの各状態に付随するSESをタスクへ詰め込む。ここで、どのようにSESをタスクに詰め込むかを説明する。状態遷移図を見ると、SES-HOOK(1～6)はフックボタン／ランプに状態に依存して処理される処理列であり、SES-RECEIVE(1～3)は電話回線からの接続要求・終了に依存して処理される処理列である。また、SES-RUSU_FUNCTIONは留守番機能ボタン／ランプの状態に依存して処理される処理列であり、このSESは４つの状態に表れている。このように、システムが作動するためのトリガーに共通性のあるものを同一のタスクにまとめる。さらに、周期起動ハンドラによって起動されるSES（SES-RECORDER_ON、SES-RECORDER_OFF）も時間に依存する共通のトリガーをもつため、同一のタスクにまとめる。

まず、上に挙げた条件でSESをタスクに詰め込み、この条件を満たさないSES（SES-DIAL_NUMBER、SES-DIAL）を最後に詰め込む。ここで、この２つのSESは接続要求中状態に付随するものである。先の条件で詰め込みが終わったタスクの接続要求中状態を調べ、これら２つのSESを詰め込める空きがあれば詰め込みを行う。しかし、これらのSESを詰め込む空きが無い場合は、新たにタスクを設ける。

この条件より、状態遷移図からタスクへの詰込み規則記述言語で記述したものを以下に示す。

・ 待機中状態

[SES-RUSU_FUNCTION] ⇒TASK1
[SES-HOOK1] ⇒TASK2
[SES-RECEIVE1] ⇒TASK3

・ 接続要求中状態

[SES-DIAL_NUMBER] ⇒TASK1
[SES-HOOK2] ⇒TASK2
[SES-DIAL] ⇒TASK3
[SES-TIMEOUT] ⇒TASK4

・ 着呼開中状態

[SES-RUSU_FUNCITON] ⇒TASK1
[SES-HOOK3] ⇒TASK2
[SES-RECEIVE2] ⇒TASK3
[SES-RECODER_ON] ⇒TASK4

・ 受話器通話中状態

[SES-RUSU_FUNCTION] ⇒TASK1
[SES-HOOK4] ⇒TASK2

・ タイムアウト状態

[SES-RUSU_FUNCTION] ⇒TASK1
[SES-HOOK5] ⇒TASK2

・ メッセージ録音中状態

[SES-HOOK6] ⇒TASK2
[SES-RECEIVE3] ⇒TASK3
[SES-RECORDER_OFF]⇒TASK4

SES モデルからタスクに詰め込まれる SES コンポーネントを決定した。SES を詰め込んだタスクのアーキテクチャを図 4.8 に示す。図で示しているタスクと SES の対応は、各状態における SES コンポーネントを SES コンポーネント名によって示している。この図に示されているものは、SES モデルとタスクの詰込み規則記述言語を Weaving によって得られる結果を示したものとなる。

TASK1	TASK2	TASK3	TASK4
case : 待機中状態	case : 待機中状態	case : 待機中状態	case : 待機中状態
SES-RUSU_FUNCTION break;	SES-HOOK1 break;	SES-RECEIVE1 break;	break;
case : 接続要求中状態	case : 接続要求中状態	case : 接続要求中状態	case : 接続要求中状態
SES-DIAL_NUMBER break;	SES-HOOK2 break;	SES-DIAL break;	SES-TIMEOUT break;
case : 着呼開状態	case : 着呼開状態	case : 着呼開状態	case : 着呼開状態
SES-RUSU_FUNCTION break;	SES-HOOK3 break;	SES-RECEIVE2 break;	SES-RECORDER_ON break;
case : 受話器通話中状態	case : 受話器通話中状態	case : 受話器通話中状態	case : 受話器通話中状態
SES-RUSU_FUNCTION break;	SES-HOO4 break;	break;	break;
case : タイムアウト状態	case : タイムアウト状態	case : タイムアウト状態	case : タイムアウト状態
SES-RUSU_FUNCTION break;	SES-HOOK5 break;	break;	break;
case : メッセージ録音中状態	case : メッセージ録音中状態	case : メッセージ録音中状態	case : メッセージ録音中状態
break;	SES-HOOK6 break;	SES-RECEIVE3 break;	SES-RECORDER_OFF break;

図 4.8　Weave 後の結果

4.7　スケジューリング

次にSESを詰め込んだタスクをどのようにスケジューリングするかを決めるため、スケジューリングのアスペクト記述言語を定義する。タスクのスケジューリングにはItIs のスケジューリングに関するシステムコールを、タスクを構成する処理列などに加えなければならない。スケジューリングを実現するシステムコールはソースコードの至る所に分散して存在するため、AOPの手法によりこれらを整理する。

今回の実装では、タスクのスケジューリングアスペクト記述言語として４つのブロックを決定している。これら４つを以下に示す。

(1) Afterブロック
(2) Eventブロック
(3) Timeブロック
(4) Handlerブロック

これより(1)と(2)については4.7.1節で、(3)と(4)については4.7.3節において説明する。

4.7.1　スケジューリングアスペクト記述言語

（1）Afterブロック

SES アプローチは連続して実行されるべき処理列を SES として記述しているので、スケジューリングに関する処理列は、SESの前か後に記述することになる。今回の実装では、SESの後にスケジューリングに関するAfterブロックを用意する。Afterブロックは以下のように定義され、[SES後の処理]にスケジューリングに関する記述を行う。SES_IDはSESを識別するためのものである。

```
SES_ID{
      After{
       [SES後の処理]
      }
}
```

このAfterブロック内にItIsのスケジューリングに関するシステムコールを決定することでタスクのスケジューリングを実現する。ここで、実際に使用したシステムコールについて説明する。スケジューリングのために使用したシステムコールは以下の２つである。これらのシステムコールについてその概要を説明する。

・**wup_tsk**(TASK_ID)

TASK_IDで示される他タスクがslp_tskの実行によって待ち状態であった場合に、その待ち状態を解除するシステムコール。

・**slp_tsk**()

自タスクを実行状態から起床待ち状態に移すシステムコール。

この２つのシステムコールを用いてAfterブロックを記述する。このAfterブロックはSESの後に位置させる。よって、SESの処理列の実行が終了した後にAfterブロックの処理列が実行されることになる。SES_IDはSES コンポーネントのSES名を意味するものである。以下に記述例を示す。

```
SES_ID{
      After{
          wup_tsk(TaskID);
          slp_tsk();
      }
}
```

wup_tsk、slp_tskのシステムコールを用いたスケジューリングの例を図4.9で説明する。この図では３つのタスクT1、T2、T3でスケジューリングを行っている。また、T1はSES1が埋め込まれており、T2ではSES2が、T3ではSES3が埋め込まれて

いるものとする。また実線の矢印は実行状態のタスクが起床待ち状態にあるタスクを解除する順序を示しており、点線の矢印はタスク内の処理の流れを示している。また、これらのタスクは同一状態において実行されているものとする。さらに、スケジューリング開始時は Task1 のみが実行状態にあり、Task2 と Task3 は起床待ち状態にあるものとする。

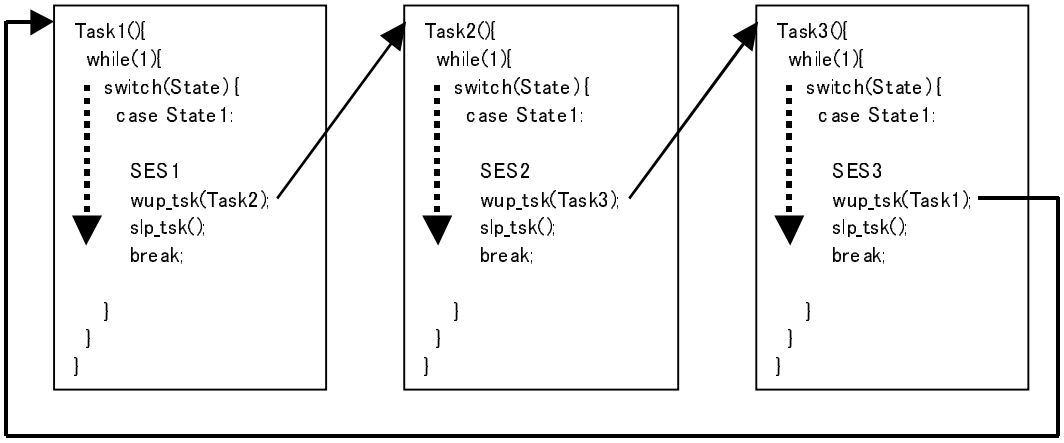


図 4.9　スケジューリングの実行例

上の例では、Task1 は SES1 を処理し終わった後に wup_tsk(Task2)のシステムコールを実行する。これにより、Task2 は実行可能状態に移る。そして Task1 が slp_tsk()を実行し、起床待ち状態に移ってから Task2 が実行状態となる。同様に Task2 が SES2 を処理し終わった後に wup_tsk(TASK3)を実行し、さらに slp_tsk()を実行することで、Task3 が実行状態に移る。このように Task1、Task2、Task3、Task1・・・という具合にこれら３つのタスクは交互にスケジューリングを行う。

（2）Event ブロック

After ブロックは SES コンポーネントの直後に位置するが、Event ブロックは SES コンポーネント内部に位置するものである。厳密には、SES コンポーネント内のイベントを出力する処理の直後に Event ブロックが与えられる。Event ブロックは以下のように定義する。SES_ID は SES を識別するためのものであり、Event_ID はイベントを識別するためのものである。

```
SES_ID{
        Event_ID{
            [イベント出力後処理]
        }
}
```

このEventブロック内にItIsのスケジューリングに関するシステムコールを決定することでタスクのスケジューリングを実現する。

4.7.2 スケジューリングアスペクト記述言語のWeave

SESのタスクへの詰込み決定後、タスクのスケジューリングに関するアスペクト記述言語を決定した。この記述に従い、システムコールを記述したAfterブロックとEventブロックをSES_IDの指定に基づき、各SESに挿入することでタスクのスケジューリングが実現される。Weaverではこの Afterブロックの挿入が行われる。図4.10では4.6節で得られたSESをタスクへの詰め込んだものとスケジューリングアスペクト記述言語をWeavingすることにより得られる結果を示している

TASK1	TASK2	TASK3	TASK4	TASK5

case : 待機中状態
 	
SES-RUSU_FUNCTION
wup_tsk(TASK2);
slp_tsk();
break;
 	
case : 接続要求中状態
 	
SES-DIAL_NUMBER
wup_tsk(TASK2);
slp_tsk();
break;
 	
case : 着呼開状態
 	
SES-RUSU_FUNCTION
wup_tsk(TASK2);
slp_tsk();
break;
 	
case : 受話器通話中状態
 	
SES-RUSU_FUNCTION
wup_tsk(TASK2);
slp_tsk();
break;
 	
case : タイムアウト状態
 	
SES-RUSU_FUNCTION
wup_tsk(TASK2);
slp_tsk();
break;
 	
case : メッセージ録音中状態
 	
wup_tsk(TASK2);
slp_tsk();
break;
	case : 待機中状態
	
	SES-HOOK1
	wup_tsk(TASK3);
	slp_tsk();
	break;
	
	case : 接続要求中状態
	
	SES-HOOK2
	wup_tsk(TASK3);
	slp_tsk();
	break;
	
	case : 着呼開状態
	
	SES-HOOK3
	wup_tsk(TASK3);
	slp_tsk();
	break;
	
	case : 受話器通話中状態
	
	SES-HOO4
	wup_tsk(TASK3);
	slp_tsk();
	break;
	
	case : タイムアウト状態
	
	SES-HOOK5
	wup_tsk(TASK3);
	slp_tsk();
	break;
	
	case : メッセージ録音中状態
	
	SES-HOOK6
	wup_tsk(TASK3);
	slp_tsk();
	break;
		case : 待機中状態
		
		SES-RECEIVE1
		wup_tsk(TASK5);
		slp_tsk();
		break;
		
		case : 接続要求中状態
		
		SES-DIAL
		wup_tsk(TASK5);
		slp_tsk();
		break;
		
		case : 着呼開状態
		
		SES-RECEIVE2
		wup_tsk(TASK5);
		slp_tsk();
		break;
		
		case : 受話器通話中状態
		
		wup_tsk(TASK5);
		slp_tsk();
		break;
		
		case : タイムアウト状態
		
		wup_tsk(TASK5);
		slp_tsk();
		break;
		
		case : メッセージ録音中状態
		
		SES-RECEIVE3
		wup_tsk(TASK5);
		slp_tsk();
		break;
			case : 待機中状態
			
			break;
			
			case : 接続要求中状態
			
			SES-TIMEOUT
			slp_tsk();
			break;
			
			case : 着呼開状態
			
			SES-RECORDER_ON
			slp_tsk();
			break;
			
			case : 受話器通話中状
			
			break;
			
			case : タイムアウト状態
			
			break;
			
			case : メッセージ録音中状態
			
			SES-RECORDER_OFF
			slp_tsk();
			break;
				case : 待機中状態
				
				KeyBoard();
				wup_tsk(TASK1);
				slp_tsk();
				break;
				
				case : 接続要求中状態
				
				KeyBoard();
				wup_tsk(TASK1);
				slp_tsk();
				break;
				
				case : 着呼開状態
				
				KeyBoard();
				wup_tsk(TASK1);
				slp_tsk();
				break;
				
				case : 受話器通話中状態
				
				KeyBoard();
				wup_tsk(TASK1);
				slp_tsk();
				break;
				
				case : タイムアウト状態
				
				KeyBoard();
				wup_tsk(TASK1);
				slp_tsk();
				break;
				
				case : メッセージ録音中状態
				
				KeyBoard();
				wup_tsk(TASK1);
				slp_tsk();
				break;

図 4.10　Weave 後の結果

4.7.3　起動ハンドラのスケジューリングアスペクト記述言語

次に、周期起動ハンドラに関するアスペクト記述言語を定義する。コードレスホンシステムでのタイマーオブジェクトによる時間的要因に従って起動するタスクは、周期起動ハンドラに用いることにより実現している。ItIs における周期起動ハンドラはタスク独立部として実行されるため、通常のタスクよりも優先して実行されるものである。周期起動ハンドラ(HANDLER_ID)の内部に Time ブロックと Handler ブロックを以下のように定義する。

```
Handler_ID{
    Time{
      [周期起動時間の指定]
    }
    Handler{
      [周期起動時間後の処理]
    }
}
```

Time ブロックでは、周期起動時間を指定する。Handler ブロックでは、この指定時間後に即座に起動すべきタスクを実行状態に移すためのシステムコールを記述する。HANDLER_ID は実装での周期起動ハンドラの関数名を意味する。以下にその記述例を示す。

```
HANDLER_ID{
        Handler{
          wup_tsk(TaskID);
        }
}
```

時間的要因に従って起動するタスクは常に起床待ち状態にしておき、この周期起動ハンドラによって起床させられることで、そのスケジューリングを実現する。そのため、周期移動ハンドラによって起床されたタスクは即座に実行権が得られるように、その他のタスクよりも優先度を高くしておかなければならない。優先度に関しては後に説明する初期起動タスクで決定する。

4.7.4 スケジューリングアスペクト記述言語の Weave

実装では 3 つの周期起動ハンドラを使用している。実装ではこれらを Timer_Handler1, Timer_Hander2, Timer_Handelr3 とした関数により実現している。これら周期起動ハンドラの概要以下に説明する。

・Timer_Hander1

発信処理を行うダイヤル時の実時間制約(ダイヤル無効:２０秒後、３０秒後、１２０秒後）に関する周期起動ハンドラ。

・Timer_Handler2

留守番機能における着信音発生から１０秒後にメッセージ録音させるための周期起動ハンドラ。

・Timer_Handler3

最長 1 分間のメッセージ録音時間を実現する周期起動ハンドラ。

周期起動ハンドラのアスペクト記述言語により、上に説明した Timer_Handler1、Timer_Handler2、Timer_Handler3 の関数内部に、スケジューリングのためのシステムコールを記述した Handler ブロックを挿入することで実時間制約を実現する。Weaver では各関数に Handler ブロックの挿入が行われる。Weave 後の周期起動ハンドラを以下に示す。

```
Timer_Handler1(){
    Time{
        20,30,120;
    }
    Handler{
        wup_tsk(TASK5_ID);
    }
}

Timer_Handler2(){
    Time{
        10;
    }
    Handler{
        wup_tsk(TASK5_ID);
    }
}

Timer_Handeler3(){
    Time{
        120;
    }
    Handler{
        wup_tsk(TASK5_ID);
    }
}
```

4.8　タスク間通信

互いに通信し合うSESコンポーネントが同一のタスク内に埋め込まれた場合、このタスク内におけるローカル変数でタスク間通信を実現することが可能となる。また、別々のタスクに詰め込まれた場合では、メールボックスやメッセージバッファを用いることで通信を実現することができる。このように互いに通信を行うSESコンポーネントに対して、タスクの詰め込み方を考慮することで、この通信実現方法が変わることとなる。そのため、タスク間通信記述言語では、オブジェクト間での通信法を具体的な指定を行わない記述法を提案する。本研究における通信の概念を図4.11に示す。

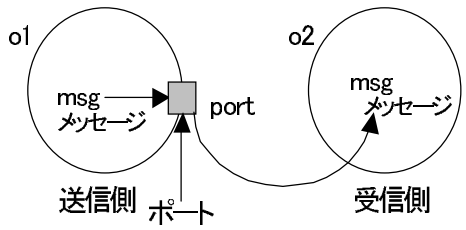


図4.11　オブジェクト間通信

図に示すように、送信側のオブジェクト(o1)はportを持ち、このportを用いてメッセージを送信する。また、受信側のオブジェクト(o2)は相手のportを指定してメッセージを受信する。よって、機能コンポーネントでは、他のオブジェクトと通信する事実のみを記述する。機能コンポーネントの記述を送信側と受信側にわけて以下に示す。

まず、送信側では送信したいメッセージを変数msgに格納し、portに出力されることを以下のようにして記述する。

```
（送信側）
   msg = ...;
   msg → o1.port
```

次に受信側では、どのオブジェクトの port からメッセージ(msg)を受け取るのかを記述し、受け取ったメッセージをどの変数に格納するのかを記述する。

```
（受信側）
   o1.port → msg;
   a = msg;
```

この記述とこれから説明するタスク間通信のアスペクト記述言語が、Weave 後にタスク間通信を実現する過程を以下で説明していく。

4.8.1 メッセージバッファ

今回、親機における受話器と本体が分離可能な電話機をモデルとして実装している。よって、相手先の電話番号を入力して発信処理を行うためには、受話器からダイヤルされたキーを本体の発信機が受信しなければならない。よって、この過程を仮想的に行うためメッセージバッファを用いて実現している。以下にメッセージバッファの概要を説明する。

・メッセージバッファ

メッセージバッファはタスク間での通信やデータ交換に使用されるものである。タスク間で通信、交換されるデータはメッセージと呼ぶ。メッセージを送信するタスクは、メッセージを格納するメモリを確保して、そのエリアに送信すべき情報を書き込み、メッセージバッファに送信する。また、メッセージバッファのバッファが一杯で十分な空き要領が無い場合には、待ち状態に入る。一方、メッセージを受信するタスクは、メッセージバッファを指定して、そこからメッセージを受信する。メッセージが到着していなかった場合には、メッセージが到着するまで待ち状態になる。

4.8.2 メッセージバッファのシステムコール

次に、Itis のメッセージバッファに関するシステムコールを説明する。これには６種類あり、以下にこれらのシステムコールの概要を説明する。

・メッセージバッファへの送信に関するシステムコール

- **snd_mbf**
  対象となるメッセージバッファにメッセージを送信する。メッセージバッファのバッファが一杯で十分な空き要領が無い場合には、待ち状態に入る。

- **psnd_mbuf**
  snd_mbf から待ち状態に入る機能を取り除いたシステムコールである。
  バッファに空き容量が無い場合は、待ち状態に入らずシステムコールを終了する。

- **tsnd_mbuf**
  snd_mbf にタイムアウト機能を付け加えたシステムコールであり、引数により待ち時間の最大値（タイムアウト値）を指定することができる。タイムアウト指定が行われた場合、待ち解除の条件が満たされないまま指定した時間が経過するとシステムコールが終了する。

  送信に関するシステムコールのパラメータには、

  ・MBUF_ID　：　送信対象となるメッセージバッファの ID
  ・msgsz　：　送信メッセージのサイズ（バイト数）
  ・message　：　送信メッセージの先頭アドレス

があり、さらに tsnd_mbf のみ以下のパラメータが加わる。

  ・TIMEOUT　：　タイムアウト指定

・メッセージバッファからの受信に関するシステムコール

- **rcv_mbf**
  対象となるメッセージバッファからメッセージを受信する。メッセージバッファを指定して、そこからメッセージを受信する。メッセージが到着していなかった場合には、メッセージが到着するまで待ち状態になる。

- **prcv_mbf**
  rcv_mbf から待ち状態に入る機能を取り除いたシステムコールである。
  バッファにメッセージが届いていない場合は、待ち状態に入らずシステムコールを終了する。

- **trcv_mbf**
  rcv_mbf にタイムアウトの機能を付け加えたシステムコールであり、引数により待ち時間の最大値（タイムアウト値）を指定するこができる。タイムアウト指定が行われた場合、待ち解除の条件が満たされないまま指定した時間が経過するとシステムコールが終了する。

  受信に関するシステムコールのパラメータには、

  ・MBUF_ID　：　受信対象となるメッセージバッファの ID
  ・message　：　受信メッセージを入れるアドレス
  ・msgsz　：　受信したメッセージのサイズ（バイト数）

  があり、さらに trcv_mbf のみ以下のパラメータが加わる。

  ・TIMEOUT　：　タイムアウト指定

4.8.3　タスク間通信のアスペクト記述言語

タスク間通信のアスペクト記述言語の構文を以下に示す。

```
Object.P_ID{
    Port_ID{
        Send{
          [メッセージを送信する処理]
        }
        Receive{
          [メッセージを受信する処理]
        }
    }
}
```

この構文は、どのオブジェクトのメソッド(Object.P_ID)がポート(Port_ID)を使ってメッセージを送受信するのかを記述するものである。送信する場合はSendブロック内の[メッセージを送信する処理]の部分に具体的な送信方法を記述する。また、受信する場合では Receive ブロック内の[メッセージを受信する処理]の部分に具体的な受信方法を記述する。

今回の実装ではメッセージバッファによりタスク間通信が実現されている。次に、このメッセージバッファで通信が行われているタスクの概要を示す。まず、受話器からのダイヤルキー情報を格納して本体発信機に送信するタスクと、送信されたダイヤルキー情報を受信し発信機により相手先に接続するタスクである。これら２つのタスクにおけるタスク間通信のアスペクト記述言語は、実際にメッセージの送受信を行う送信側オブジェクトと受信側オブジェクトに分けて定義する。また上に挙げたように送信・受信に対して複数のシステムコールが存在するため、どのシステムコールを使用するかを選択する必要がある。

タスク間通信を行うのは、SES コンポーネント内の個々の処理である。そのため、処理列を構成している個々のオブジェクトの処理に、通信を実現する手法を確立する必要がある。そこでこの手法を説明するにために、送信側のオブジェクトを o1 とし、そのメソッドを p1 とする。また、受信側のオブジェクトを o2 とすし、そのメソッドを p2 とする。メッセージの送受信を行うために用いるポートは Port とする。

まず送信側のアスペクト記述言語について説明する。送信側では以下に示す(A)のように記述を行う。

(A)

```
o1.p1{
  o1.Port{
        Send{
              [メッセージを送信する処理]
        }
  }
}
```

(A)では、オブジェクト(o1)の送信するための処理(p1)が Send ブロックを持つことを示している。ItIs ではメッセージバッファでの送信に関するシステムコールが３種類ある。よって、これらのシステムコールを選択する必要がある。以下の(B)には、メッセージバッファによる送信処理として選択可能なものを３つ示している。

(B)

```
snd_mbf(MBUF_ID, msg, sizeof(msg));
psnd_mbf(MBUF_ID, msg, sizeof(msg));
tsnd_mbf(MBUF_ID, msg, sizeof(msg), TIMEOUT);
```

次に受信側について説明する。受信側では以下の(C)に示す記述を行う。

(C)

```
o2{
  o1.Port{
        Receive{
              [メッセージを受信する処理]
        }
  }
}
```

(C)では、オブジェクト(o2)の受信するための処理(p2)が Receive ブロックを持つことを示している。ItIs ではメッセージバッファでの受信に関するシステムコールは３種類ある。よって、これらのシステムコールを選択する必要がある。以下の(D)

には、メッセージバッファによる受信処理として選択可能なものを３つ示している。

(D)

```
rcv_mbf(msg, msgsz, MBUF_ID);
prcv_mbf(msg, msgsz, MBUF_ID);
trcv_mbf(msg, msgsz, MBUF_ID, TIMEOUT);
```

4.8.4　タスク間通信のアスペクト記述言語の Weave

タスク間通信のアスペクト記述言語とSESコンポーネントがWeaveする過程の例を、実装で通信を行う部分で説明する。

まず、送信側から説明する。メッセージバッファへ送信している SES は SES-DIAL_NUMBER である。この SES-DIAL_NUMBER を例として Weave の流れを説明する。SES-DIAL_NUMBER で通信を行う処理は図 3.14 の o7.p4：KeySend である。ここで、SES-DIAL_NUMBER での送信を行うオブジェクト、メッセージバッファのシステムコール、送信対象となるメッセージバッファ、送信するメッセージを代入する変数を以下に示す。

・オブジェクト ： o7(ダイヤルボタン／ランプ オブジェクト)
・システムコール ： snd_mbf
・送信対象メッセージバッファ ： MBUF1_ID
・送信メッセージを代入する変数: one_digit

これらの情報から、タスク関通信アスペクト記述言語は以下の図 4.11 になる。

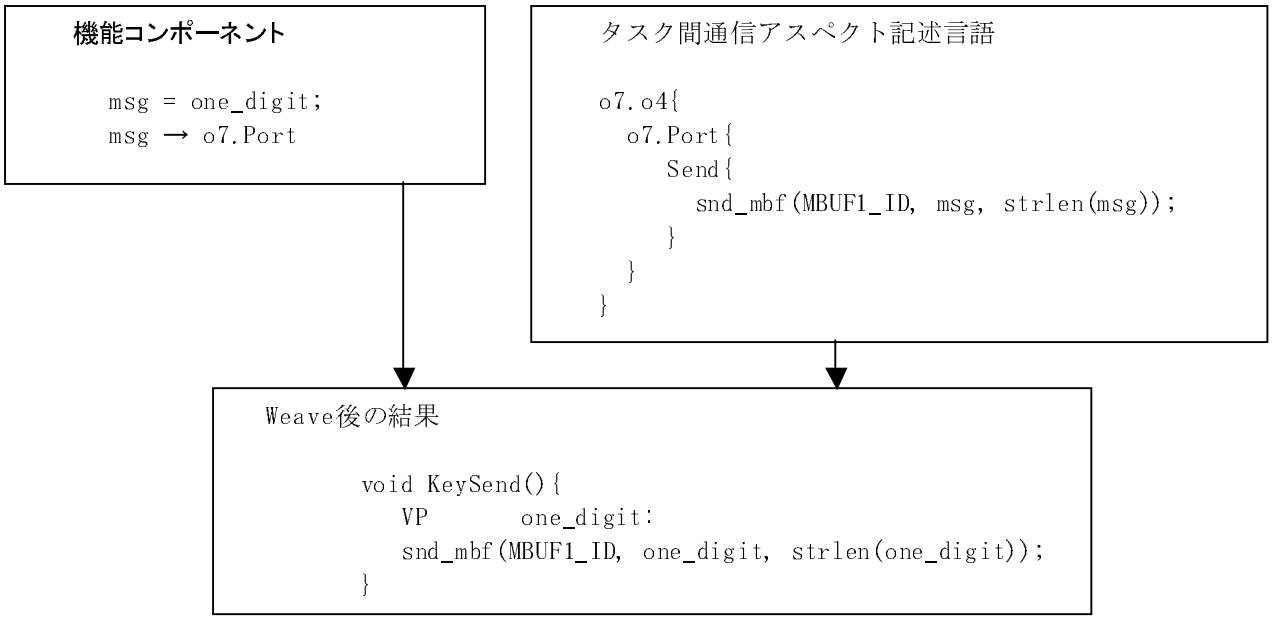


図 4.11 送信側の Weave の流れ

Weave では、機能コンポーネントで指定した one_digit をタスク間通信アスペクト記述言語で指定した msg の部分に置き換える。

次に、受信側について説明する。実装では、メッセージバッファから受信している SES は SES-DIAL である。この SES-DIAL を例として Weave の流れを説明する。SES-DIAL で通信を行う処理は図 3.5.13 の o11.p1：KeyReceive である。ここで、SES-DIAL での受信を行うオブジェクト、メッセージバッファのシステムコール、受信対象となるメッセージバッファ、メッセージを受信するための変数を以下に示す。

・オブジェクト ： o11(発信機 オブジェクト)
・メソッド ： o11.p1(KeyReceive)
・システムコール ： prcv_mbf
・受信対象メッセージバッファ ： MBUF1_ID
・受信メッセージを格納する変数.： msg_num[1]

これらの情報から、タスク関通信アスペクト記述言語は以下の図 4.12 になる。

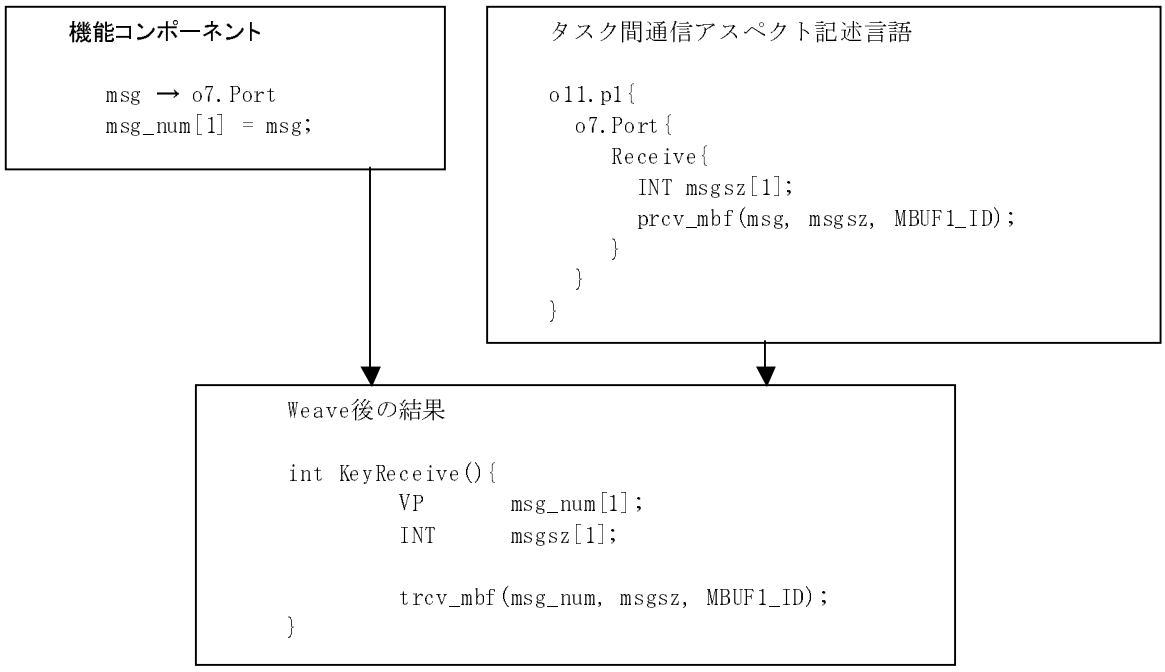


図 4.12

Weave では、機能コンポーネントで指定した msg_num[1]をタスク間通信アスペクト記述言語で指定した msg の部分に置き換える。

4.9　ItIs の初期起動タスクについて

ItIs では初期起動タスクからすべての実行が開始される。これは C 言語のメイン関数のようなものであり、本研究での初期起動タスク(Initial_Task)の位置付けを説明する。

まず、今回の実装では図 4.13 に示すように、タスクの数は全部で６つ存在する。

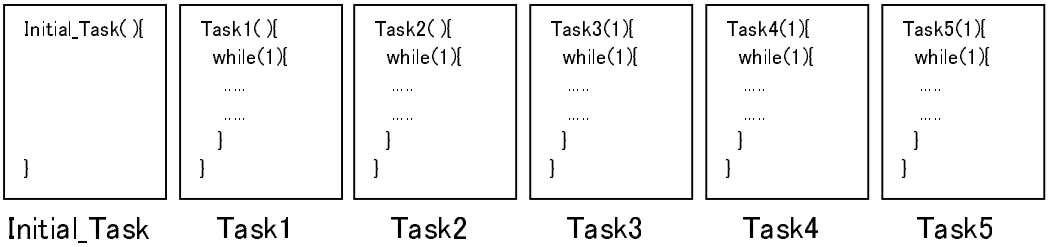


図 4.13　今回のタスク実装例

しかし、初期起動タスクもその他のタスクと同様のものであり、上の図で示している Task1 の処理部を Initial_Task に記述することで、図 4.14 のように実装し、タスク数を減らすことも可能である。しかし、今回の実装では、この図に示すような実装は行っていない。

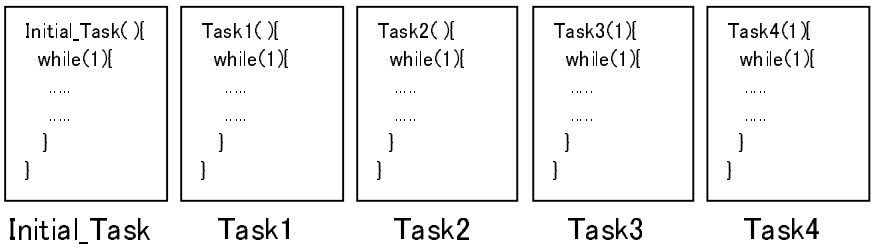


図 4.14　他に考えられるタスク実装例

初期起動タスク以外のタスクを用いる場合は、そのタスクを新たに生成しなければならない。これは ItIs のタスクの生成に関するシステムコールを用いることで実現される。メッセージバッファ、周期起動ハンドラなどを使用する場合も、同じように

システムコールを用いてこれらを生成しなければならない。さらに、プログラムの実行を終了する場合、生成したものに関しては必ず削除して実行を終えなければならない。そのため、初期起動タスクではあらかじめこれらを生成・削除するシステムコールを記述する箇所としている。また、初期起動タスクにはプログラムの実行が確実に終了できるように最高優先度のプライオリティを与えている。

実装した初期起動タスクのプログラムを以下に示し、この初期起動タスクに関して Weave する方法を説明する。また、ここで示すコードは実装したコードの一部を省略したものである。

```
void Initial_Task(INT stacd){
        chg_pri(TSK_SELF, HIGH_PRIORITY);  ──────────▶(1)
        create_start_task(TASK1_ID, &task1, LOW_PRIORITY, STKSIZE, 1);
        create_start_task(TASK4_ID, &task4, LOW_PRIORITY, STKSIZE, 4);  } (2)
        create_start_task(TASK5_ID, &task5, MID_PRIORITY, STKSIZE, 5);
        create_start_mbuf(MBUF1_ID, &mbuf1, BUFSIZE, MAXSIZE);
        create_start_cych1(CYC1_ID, CACT, CTIME1);
        create_ start_cych1(CYC1_ID, CACT, CTIME2);
        create_start_cych1(CYC1_ID, CACT, CTIME3);
        tslp_tsk(120000);  ──────────────────────────▶(3)
        terminate_delete_task(TASK1_ID);
        terminate_delete_task(TASK4_ID);
        terminate_delete_task(TASK5_ID);  } (4)
        terminate_delete_mbuf(MBUF1_ID);
        terminate_delete_cych(CYC1_ID);
        FINISH();
}
```

(1) 初期起動タスクの優先度に関する記述

初期起動タスクに最高優先度を与えるシステムコールを記述している。

(2) 生成と起動に関する記述

create で始まる関数がタスク、メッセージバッファ、周期起動ハンドラの生成と起動に関する関数である。

(3) スケジューリングに関する記述

初期起動タスクのスケジューリングに関するシステムコールが記述されており、tslp_tsk は引数に時間を指定し、指定した時間だけ自タスクが起床まち状態に移るものである。

(4) 終了と削除に関する記述

terminate で始まる関数は生成したタスク、メッセージバッファ、周期起動ハンドラを正常終了させ、削除するための関数である。また、FINISH()は ItIs を正常終了するために必要な関数である。

今回の実装では、プログラムはある一定時間起動後に終了するよにしている。永久に実行したい場合、(3)で記述したスケジューリングに関するシステムコールを slp_tsk にし、初期起動タスクを永久に起床待ち状態に移すことで実現される。その場合(4)での終了と削除に関する記述は必要ない。

上に挙げた初期起動タスクは Weave 後に得られるものである。よって Weave 前の初期起動タスクについて説明し、上に上げた初期起動タスクが得られる過程を説明する。

初期起動タスクの生成・起動、さらに終了・削除の関数に対して、ライブラリと機能コンポーネントに分割しておく。これらの関数でパターン化される部分をライブラリに、また関数内部に現れるシステムコールのエラー処理を機能コンポーネントに分割する。ここでは、初期起動タスクの(2)に該当するタスク生成・起動を行う関数と(4)に該当するタスク終了・削除の関数を例にして説明する。create_start_task()と terminate_delete_tast()の内部を以下に示し、まず create_start_task()と terminate_delete_task()についての概要を説明する。

```
ER create_start_task(ID taskid, FP task, PRI tskpri, INT stksz, INT stacd)
{
        t_CTSK  ctsk;
        ctsk.exinf = 0;
        ctsk.tskatr = TA_HLNG;          } タスクの属性
        ctsk.itskpri = tskpri;
        ctsk.stksz = stksz;

        cre_tsk(tskid, &ctsk);  ←（タスクを生成するシステムコール）
        if(ercd != E_OK){
            syslog(LOG_NOTICE, “error %d with cre_tsk(%d)”, ercd, tskid);  } エラー処理
            return(ercd);
        }

        sta_tsk(tskid, stacd);  ←（タスクを起動するシステムコール）
        if(ercd != E_OK){
            syslog(LOG_NOTICE, “error %d with sta_tsk(%d)”, ercd, tskid);  } エラー処理
            return(ercd);
        }
}
```

```
ER terminate_delete_task(ID tskid)
{
         ercd = ter_tsk(tskid);  ←（ タスクを終了させるシステムコール ）
         if(ercd != E_OK){
            syslog(LOG_NOTICE, “error %d with ter_tsk(%d)”, ercd, tskid);  } エラー処理
            return(ercd);
         }

         ercd = del_tsk(tskid);  ←（ タスクを削除するシステムコール ）
         if(ercd != E_OK){
            syslog(LOG_NOTICE, “error %d with del_tsk(%d)”, ercd, tskid);  } エラー処理
            return(ercd);
         }
}
```

・ create_start_task()

タスクの属性を決定する部分と、タスクの生成・起動に関するシステムコールの部分からなる。さらにこれらのシステムコールにエラー処理を加えている。タスクの属性の部分では生成するタスクに対してタスク起動優先度、スタックサイズといった情報を引数で与えてやることにより、その初期設定を行っている。tskid で指定された

ID番号を持つタスクを生成し、起動している。

・ terminate_delete_task()

タスクの終了と削除を行う２つのシステムコールからできている。同様にこれらのシステムコールにもエラー処理を加えている。ここでの引数は１つしかなく、tskidで指定されたタスクに対して終了・削除が行われる。

このように、メッセージバッファ、周期起動ハンドラに関する生成・起動、終了・削除の関数もここで説明したタスクと同じような概要になっており、生成・起動に関しては属性を引数で与えてから行い、終了・削除に関してはこれらの対象となる IDを引数として与えてやればよい。そのため、初期起動タスクに記述した生成・起動、終了・削除の関数はパターン化が可能である。そこで、これらをライブラリに格納する。そして、機能コンポーネントにはシステムコールのエラー処理を書く。

よって、Weaver ではライブラリにある個々のシステムコールに対して、機能コンポーネントに書かれたエラー処理を適切に割り当てることが行われる。

4.10　実行可能なプログラムまでの過程

タスクの詰込み規則記述言語に始まり、スケジューリングアスペクト記述言語、タスク間通信アスペクト記述言語と初期起動タスクの流れでWeaveを行い、実行可能なプログラムまでの過程を示してきた。

AOPではコンポーネントプログラムとアスペクトプログラムにより対象システムを記述する。さらにこれらのプログラムはWeaverと呼ばれる合成器を用いて合成することにより、実行可能なプログラムを獲得する。今回の研究ではWeaveの部分を手動で行うことで、実行可能なプログラムを得ている。

実行可能なプログラムを獲得するには、図4.15に示しているように、SESモデル、SESコンポーネント、機能コンポーネント、タスクへの詰め込み規則記述言語、スケジューリングアスペクト記述言語、タスク間通信記述言語のすべてがWeaverによって合成され、実行可能なプログラムを得ることができる。

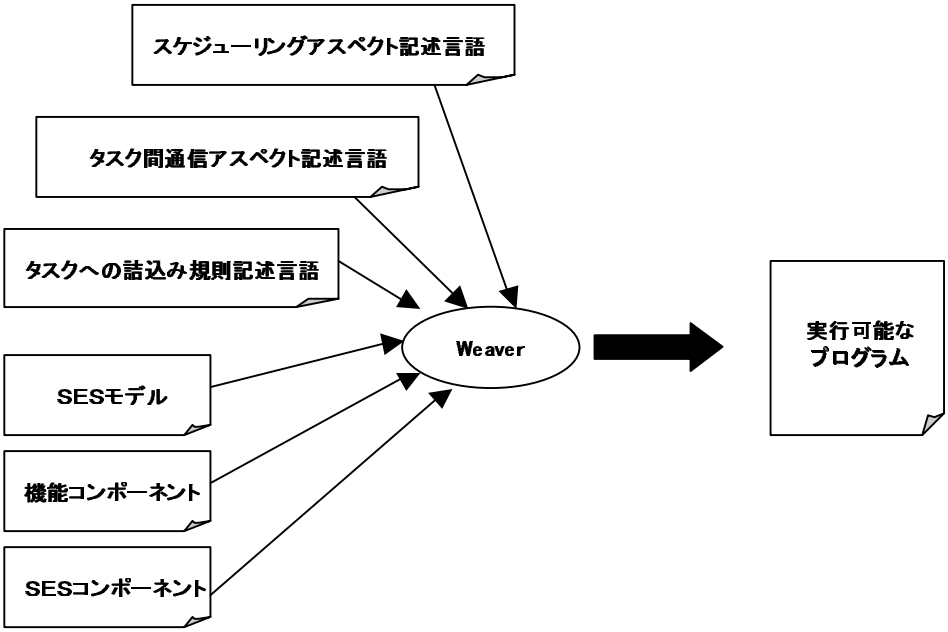


図 4.15　実行可能なプログラムを得る過程

第５章

考 察

今回、第４章でアスペクト指向プログラミングにより電話機システムの実装を行った。これを実現するために、タスクの詰め込み記述言語、スケジューリングアスペクト記述言語、さらにタスク間通信アスペクト記述言語という流れによって、実行可能なソフトウェアを獲得する過程を説明した。また４章で決定したタスクの詰込み規則記述言語、スケジューリングアスペクト記述言語、タスク間通信アスペクト記述言語の内容が、これら３つのアスペクトの間でどのように影響を及ぼし合い、またどんな関係を持ちえるのかについて考察を進めていく。

5.1　タスクの詰込みとスケジューリングの関係

電話機システムにおける時間的制約を実現するために、周期起動ハンドラを用いることで実時間制約に関する問題を解決することを試みた。周期起動ハンドラにはスケジューリングに関する Handler ブロックを用意し、その内部に、起床待ち状態にあるタスクに対して、起床待ち解除する wup_tsk システムコールを記述した。しかし、これだけでは周期起動ハンドラを用いることで、実時間制約を満たすことはできない。実時間制約を満たすためには、この周期起動ハンドラによって wup_tsk が実行されることで起床待ち解除されるタスクが、解除されその時点で確実に実行を開始しなければならない。これを実現するためには、周期起動ハンドラによって起床待ち解除されるタスクが他のタスクよりも高い優先度をもつ必要がある。よって、時間的要因によって起動される SES コンポーネントは１つのタスク内にまとめ、このタスクの優先度を他のタスクの優先度よりも高く設定することで、実時間制約を満足するスケジューリングを実現することができる。

今回の実時間制約を満たすためのスケジューリング方法では、上で説明したスケジューリング法からもわかるように、時間的要因に従って起動する SES コンポーネン

トは優先度の高いタスクに埋め込まなければならないという、タスクの詰め込みに関する制約が生じてしまうことになる。よって、タスクの詰め込み規則記述言語により SES をタスクへ詰め込む際に、実時間制約を満たすためには、このタスクの詰め込みに関する制約を考慮して詰め込まなければならないことになる。よって、この制約のために SES のタスクへ詰め込みに対する自由度が低くなってしまう。このようなタスクへの詰め込みに関する制約を無くすためには、スケジューリングを行うタスクは、すべて同一の優先度をもたなければならないことになる。

よって、タスクの詰め込みに関して、このような制約を生じさせないためには、スケジューリングを行うすべてのタスクに同一の優先度を持たせなければならないことになる。このようにタスクに対して同一の優先度を持たせるということは、スケジューリングに関しての制約を生じさせることになる。今回の実装では、wup_tsk、と slp_tsk の２つのシステムコールを用いることでタスクが交互に実行できるスケジューリングを実現している。この２つのシステムコールを用いてスケジューリングを実現する場合では、タスクの優先度に制約は働かない。その例を図 5.1 に示して説明する。

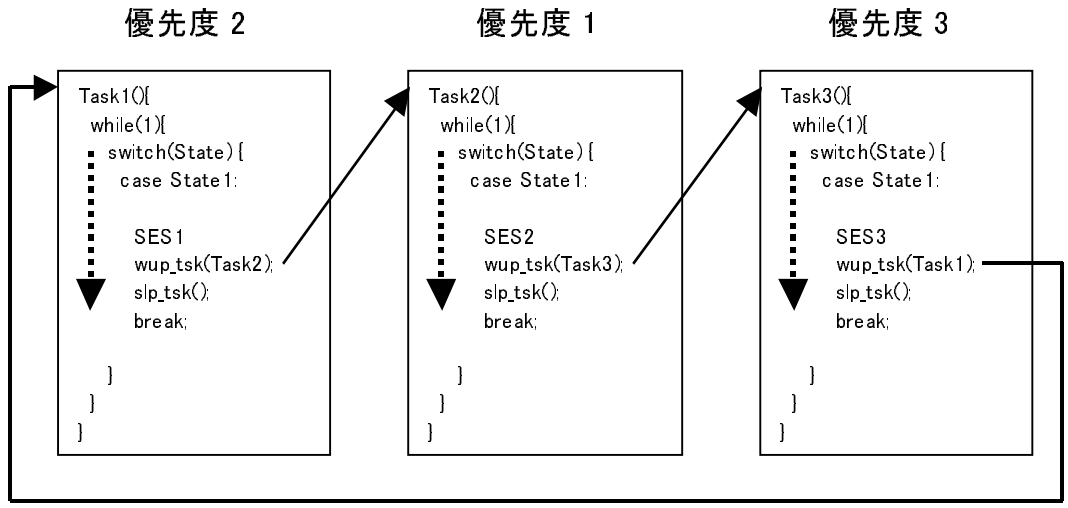


図 5.1　スケジューリングの例

ここでは、優先度１が最高優先度であるとする。この図では、３つのタスク T1、T2、T3 でスケジューリングを行っている。また、T1 は SES1 が埋め込まれており、T2 では SES2 が、T3 では SES3 が埋め込まれているものとする。また実線の矢印は実行

状態のタスクが起床待ち状態にあるタスクを解除する順序を示しており、点線の矢印はタスク内の処理の流れを示している。また、これらのタスクは同一状態において実行されているものとする。さらに、スケジューリング開始時は Task1 のみが実行状態にあり、Task2 と Task3 は起床待ち状態にあるものとする。

上の例では、Task1 は SES1 を処理し終わった後に wup_tsk(Task2)のシステムコールを実行する。これにより、Task2 は実行可能状態に移る。そして Task1 が slp_tsk()を実行し、起床待ち状態に移ってから Task2 が実行状態となる。同様に Task2 が SES2 を処理し終わった後に wup_tsk(TASK3)を実行し、さらに slp_tsk()を実行することで、Task3 が実行状態に移る。このように Task1、Task2、Task3、Task1・・・という具合にこれら３つのタスクは交互にスケジューリングを行う。

このように TASK1 が実行中のとき、TASK2 と TASK3 は起床待ち状態となる。また、TASK2 が実行中のときは TASK1、TASK3 が起床待ち状態、TASK3 が実行中のときは、TASK1、TASK2 が起床待ち状態となる。この図からもわかるように優先度の違いによるスケジューリングの影響はない。

タスクの優先度を全て同一にした状態で、スケジューリング関する制約が生じる例を説明する。ItIs におけるスケジューリングを実現するシステムコールは wup_tsk と slp_tsk だけではない。chg_tsk というシステムコールを使うことでスケジューリングを行うことが可能である。このシステムコールについて以下で説明する。

・ chg_tsk(TsakID, Priority);
　タスクの優先度を変更するためのシステムコール。
　TaskID で示されたタスクの現在の優先度を Priority で示される値に変更する。

このシステムコールを用いてタスクの優先度を変化させることで、タスクのスケジューリングを実現することができる。

以上のことをまとめると、タスクの詰め込みに関する制約を無くすために、タスクに対して全て同一の優先度を持たせることは、スケジューリングを実現するためのシステムコールに制約を生じさせることになる。このように、タスクの詰め込み規則記述言語とスケジューリングアスペクト記述言語の間にトレードオフが生じることがわかる。タスクの詰め込み規則記述言語に自由度をもたせると、スケジューリングアスペクト記述言語に対する自由度が下がる。

逆に、スケジューリングアスペクト記述言語に自由度を持たせると、タスクの詰め込み規則記述言語に対しての自由度は下がる。

しかし、実時間制約のあるSESコンポーネントを時間的要因に従って的確に実行させるためには、全てのタスクを同一の優先度として実現するのは不可能である。この問題を回避するためには時間的要因に従って実行するSESコンポーネントのタスクを一時的に周りのタスクより優先度を高くする必要がある。次の節でこの実現方法を取り上げる。

5.2 スケジューリング法の変更

今回の実装でのタスクの詰め込み方を変更せずに、周期起動ハンドラによるラウンドロビン方式のスケジューリングに変更する。この周期起動ハンドラがどのように実現されているかについては、以降で述べる。新たなラウンドロビン方式のスケジューリングにより各SESコンポーネントに対してAfterブロックを用意する必要がなくなる。よって、以下の図5.2に示すように、ラウンドロビンでスケジューリングを行うために、全てのタスクを同一優先度（LOW_PRIORITY）としている。しかし、これでは時間的な要因により実行されるSESコンポーネントが、実時間制約を満たすように実行される保証がない。また、下の図にある接続要求中状態では、SES-TIMEOUTは時間的な要因によって起動されるものである。

TASK1	TASK2	TASK3	TASK4	TASK5
case：待機中状態	case：待機中状態	case：待機中状態	case：待機中状態	case：待機中状態
SES-RUSU_FUNCTION	SES-HOOK1	SES-RECEIVE1		KeyBoard();
break;	break;	break;	break;	break;
case：接続要求中状態	case：接続要求中状態	case：接続要求中状態	case：接続要求中状態	case：接続要求中状態
SES-DIAL_NUMBER	SES-HOOK2	SES-DIAL	SES-TIMEOUT	KeyBoard();
break;	break;	break;	break;	break;
case：着呼開状態	case：着呼開状態	case：着呼開状態	case：着呼開状態	case：着呼開状態
SES-RUSU_FUNCTION	SES-HOOK3	SES-RECEIVE2	SES-RECORDER_ON	KeyBoard();
break;	break;	break;	break;	break;
case：受話器通話中状態	case：受話器通話中状態	case：受話器通話中状態	case：受話器通話中状	case：受話器通話中状態
SES-RUSU_FUNCTION	SES-HOO4			KeyBoard();
break;	break;	break;	break;	break;
case：タイムアウト状態	case：タイムアウト状態	case：タイムアウト状態	case：タイムアウト状態	case：タイムアウト状態
SES-RUSU_FUNCTION	SES-HOOK5			KeyBoard();
break;	break;	break;	break;	break;
case：メッセージ録音中状態	case：メッセージ録音中状態	case：メッセージ録音中状態	case：メッセージ録音中状態	case：メッセージ録音中状態
	SES-HOOK6	SES-RECEIVE3	SES-RECORDER_OFF	KeyBoard();
break;	break;	break;	break;	break;

図 5.2　Weave 後の結果

ここで、改めて SES-TIMEOUT について述べる。
SES-TIMEOUT は発信処理（相手先に電話をかける）を行うときに実行される可能性があるものである。
この SES-TIMEOUT が実行されるのは、電話番号入力時において、

・最初のダイヤルキーは３０秒以内に押されなければならない。
・２つ目以降のダイヤルキーは２０秒以内に押されなければならない。
・相手先の電話番号は２分以内に入力し終わらなければならない。

という、この３つのいずれかの時間的制約が満たされなかった場合に、SES-TIMEOUT が実行される。

図 5.2 の接続要求中状態を見てもわかるように、ラウンドロビン方式により、このままでは時間的要因に関わらず SES-TIMEOUT が実行されてしまう。また、SES-TIMEOUT だけでなく、SES-RECORDER_ON、SES-RECORDER_ON についても時間的要因により実行されるものなので、同様のことが言える。

よって、新たにスケジューリングアスペクト記述言語として、ここに Before ブロックを用意する。この Before ブロックの中にスケジューリングに関するシステムコールを記述し、これを時間的な要因によって実行される SES コンポーネントの頭に埋め込む。

```
SES {
        Before{
            slp_tsk();
        }
        [SESコンポーネント]
}
```

ここで時間的な要因によって実行される SES コンポーネントは、SES-TIMEOUT、SES-RECORDER_ON、SES-RECORDER_OFF である。この 3 つの SES コンポーネントの前に Before ブロックを埋め込む。よって、これらは slp_tsk によって起床待ち状態に移される。ここまでの過程では、時間的要因に関わる SES コンポーネント以外の SES をラウンドロビン方式で実行可能にしたまでにすぎない。これでは、時間的要因に関わる SES コンポーネントが実行されないままに終わってしまう。よって、ここでも新たな周期起動ハンドラを用いることで、実時間制約を満たせるようにする。周期起動ハンドラでは、slp_tsk によって起床待ち状態にあるタスク（SES-TIMEOUT、SES-RECORDER_ON、SES-RECORDER_OFF が埋め込まれている TASK4）を wup_tsk により起床させる。しかし、起床待ちを解除した SES コンポーネントは、即時に実行されるわけではないため、起床させると同時に時間的な要因により実行される SES コンポーネントが詰め込まれているタスクの優先度を一時的に高くする。これには chg_pri システムコールを用いる。以上のことから、周期起動ハンドラのアスペクト記述言語は以下のようになる。

```
HANDLER_ID{
      Handler{
          chg_pri(TASK4, MID_PRIORITY);
          wup_tsk(TASK4);
       }
}
```

以上により、時間的に実行される TASK4 は一時的に優先度が高くなるが、これを再び LOW_PRIORITY の優先度に戻さなければ、ラウンドロビンスケジューリングが行われない。よって、ここで、スケジューリングアスペクト記述言語である After ブロックを用意する。この After ブロック内で一時的に高くした優先度をもとの優先度に戻すためのシステムコールを記述する。これにも chg_pri を用いる。よって After ブロックは以下のようになる。

```
SES_ID {
          After{
          chg_pri(TASK4, LOW_PRIORITY);
          }
}
```

ここで、今回のラウンドロビンのスケジューリングを実現している周期起動ハンドラを説明する。このスケジューリングを実現する周期起動ハンドラのスケジューリングアスペクト記述言語を以下に示す。

```
HANDLER_ID{
      Time{
          0.01
      }
      Handler{
          rot_rdq(LOW_PRIORITY);
      }
}
```

ここで使用している。rot_rdq システムコールについて説明する。

- rot_rdq(tskpri)
  tskpri で示されている優先度のレディーキューを回転する。即ち、その優先度のレディーキューにつながれているタスクをレディーキューの最後尾につなぎかえ、同一優先度のタスクの実行を切りかえる。このシステムコールを一定時間間隔で発行することによりラウンドロビン・スケジューリングを行うことが可能となる。

この rot_rdq システムコールを周期起動ハンドラから一定時間間隔で呼び出すことにより、同一優先度(LOW_PRIORITY)のタスクをラウンドロビン・スケジューリングを行っている。

ここまでの、Before ブロック、After ブロックを詰め込んだ Task4 の結果と周期起動ハンドラを図 5.3 に示す。

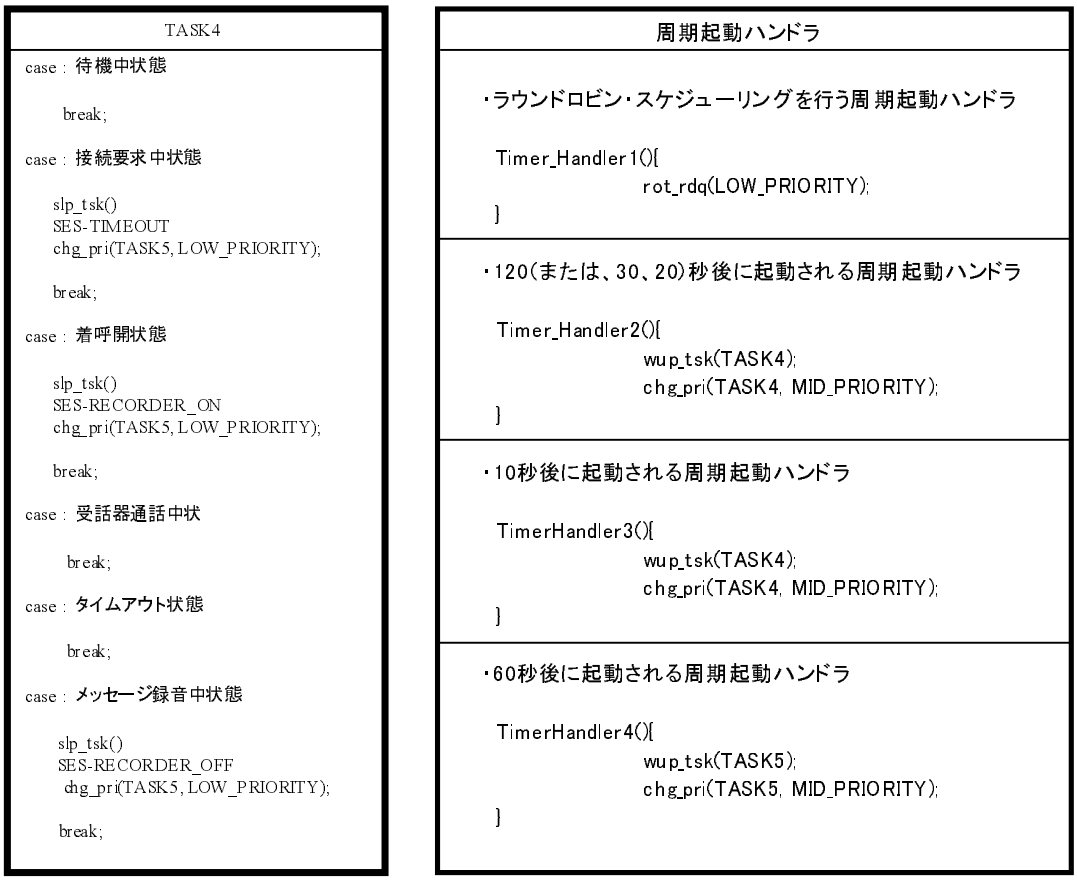


図 5.3　Weave 後の結果

この図に示すように、タスクへの詰め込み方を変えずにスケジューリング法だけを変更したが、このスケジューリング法によっても実行可能なプログラムが得られた。このラウンドロビンにおけるスケジューリング法では、新たにスケジューリング記述言語として Before ブロックを定義することで実現が可能となった。この他にもスケジューリング法は考えられる。タスクの詰め込み方を全面的に変更することも可能であり、変更によってスケジューリング法に影響することが予測される。

今回では、最初にタスクの詰め込み方を決定した後にスケジューリング法を決定したが、スケジューリング法を最初に決定することで、タスクの詰め込み方にも影響することが予測できる。

5.3 タスクの詰め込みとタスク間通信の関係

今回の実装では、タスク間通信を実現するためにメッセージバッファを用いることで実現した。タスク間通信を行っていた SES コンポーネントは SES_DIAL と SES_RECEIVE であった。これら２つのコンポーネントが出現する SES モデルの一部を図 5.4 に示す。

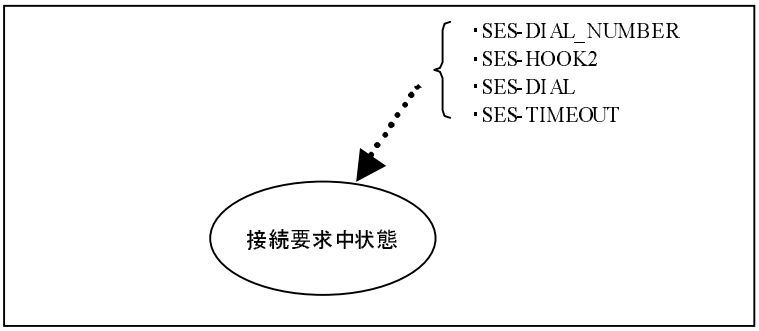


図 5.4　SES モデルの一部

図からもわかるように、これら２つの SES コンポーネントは SES モデルにおける状態遷移図において同一の状態に付随している。よって、この２つの SES コンポーネントは別々のタスクに詰め込まれることになる。今回のメッセージバッファを用いたのは、受話器と本体がコードレスである電話機を仮想的に実現するためであった。このように別々のタスクに通信する SES コンポーネントが埋め込まれている場合、タスク間通信のシステムコールを用いて行うことになる。

また、互いに通信し合うSESコンポーネントが同一のタスク内に埋め込まれた場合、このタスク内におけるローカル変数でタスク間通信を実現することが可能となる。

このように互いに通信を行うSESコンポーネントに対して、タスクの詰め込み方を考慮することで、この通信実現方法が変わることとなる。

第６章

まとめ

６．１　まとめ

本研究ではSESアプローチに基づく実時間制約を考慮した組み込みシステムの実装法について研究を行った。また、実装にはAOPの手法により、SESのタスクへの詰め込み方、タスクのスケジューリング、タスク間通信に着目し、これらをアスペクトとして捕らえた。さらに、それぞれの記述言語を決定し、最終的な実行可能なプログラムを獲得するまでの過程を示した。今回の実装により、ある程度の規模のシステムが提案した手法により記述可能なことを確認した。また、最終的なプログラムを、SESモデル、SESコンポーネント、機能コンポーネントと３つの記述（タスクへの詰め込み記述、スケジューリング記述、タスク間通信記述）により、完全に定義することができた。

６．２　今後の課題

・タスク間通信ではメッセージバッファに関するアスペクト記述言語を示した。しかし、ItIsにはこの他にメールボックス、セマフォ、イベントフラグといった、タスク間通信を実現するシステムコールがある。これらに関するアスペクト記述言語を決定し、他のアスペクトとの関係を調べる。

・今回、実行可能なプログラムを得るために手動によるWeaverでこれを実現している。よって自動的に合成を実現し、最終的なプログラムが得られるWeaverを実現する。