JAIST Repository
https://dspace.jaist.ac.jp/

Title	SESモデルに基づいた組み込みプログラム自動生成システムの研究
Author(s)	古城，敬章
Citation	
Issue Date	2001-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1473
Rights	
Description	Supervisor:片山 卓也，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

SES モデルに基づいた組み込みプログラム自動生成システムの研究

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

古城 敬章

平成 13 年 2 月 15 日

Copyright © 2001 by Takaaki Koshiro

目 次

第 1 章

はじめに

1.1 研究の背景

現在、ソフトウェア開発手法としてオブジェクト指向方法論が広く使われている。オブジェクト指向方法論は、ソフトウェアの柔軟性、再利用性を高める方法として現在最も進んだアプローチである。しかし、電話器などの組み込みソフトウェア開発においては、実時間制約やハードウェア構成などの非機能的要件と呼ばれる組み込むシステム特有の厳しい制約がある。これらの制約は、設計工程において考慮されるが、現在提案されているオブジェクト指向方法論では分析工程中心に考えられており、どのように扱うかを明確にしていないため適切に扱うことができない。

そこで、本研究では青木により提案されたこれらの制約を扱うのに適した SES モデル [?] に基づき、これらの制約を満たすプログラムの自動生成を行うシステムの提案を行う。

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 モデルを用いる。本研究では、電話器を例題として SES モデルに基づく実装をどこまで自動化できるかについての研究を行う。実装に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リアルタイムオペレーティングシステム (以下、RTOS と略す) を用いる。これは、SES モデルは RTOS に系統的に変換することができる性質を活かすためであるというのと、実際の組み込みシステムにおいても RTOS が多用されているためでもある。

森本の研究により SES モデルを用いた組み込みシステム開発でおいてタスクへの詰め

込み、スケジューリング、タスク間通信の３つを決定すべきことが分かった。タスクへの詰め込みとは、 SES モデルは SES と呼ばれるものを処理単位とし、一方、RTOS はタスクを処理単位としているために、SES をタスクにマッピングすることを意味している。スケジューリングは SES の実行順序の決定、タスク間通信はタスクに詰め込まれた SES が他の SES と通信する方法の決定である。

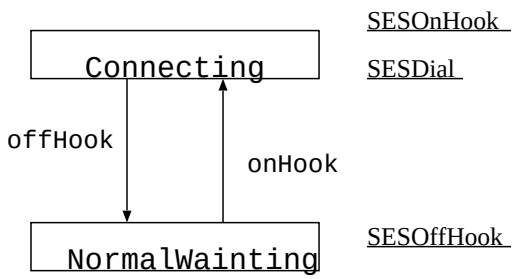
本研究ではスケジューリングを中心にこれらの要件の記述をもとに、RTOS 上で動作するプログラムをどこまで自動生成できるかについて研究を行う。また、スケジューリングに関する記述を求める手法についても考察を行う。

第 2 章

SES モデル

2.1 SES モデルの概要

SES モデルは、オブジェクト指向組み込みシステム開発手法である。SES モデルでは、SES(synchronus execution sequence) と呼ばれる同期した処理を単位とし、この SES を用いて対象システム全体の振舞を記述する。



- 処理
  個々の処理はオブジェクトによって所有される。オブジェクト**o-name**が処理**p-name**を持つとき、**o-name.p-name** と記す。

- 分岐
  菱形で表現し、条件分岐を行うあめの条件は菱形のなかに記述する。分岐には実線矢印と破線矢印があり、条件が成立しているときは実線矢印の先に示される処理列が実行され、条件が不成立の場合は破線矢印の先の処理列が実行される。また、条件成立したときや条件が不成立したときに、何も実行しないときは矢印を省略する。

- イベント
  SES の実行を制御するために用いる。

2.2 SES モデルの実装

SES モデルには、SES モデルで記述されたシステムを RTOS 上に実装するためのテンプレート、リファレンスアーキテクチャがある。リファレンスアーキテクチャは、SES コンポーネントと機能コンポーネントにより構成されている。

2.2.1 リファレンスアーキテクチャ

リファレンスアーキテクチャ全体の構造を図??に示す。

状態マネージャー

システムの現在状態を管理するもので、現在状態を獲得する `get_State()` 関数と `change_State()` 関数の 2 つの関数と、現在状態を表す変数 `current_state` によって構成されている。関数 `current_state` は SES におけるイベント出力の実装として使用される。また、各々の状態は整数によって表現され、マクロとして定義される。

機能コンポーネント

SES モデルに出現するオブジェクトは機能コンポーネントとして実装される。この機能コンポーネントは C 言語の変数と関数で構成され、オブジェクトがもつ処理は関数とし

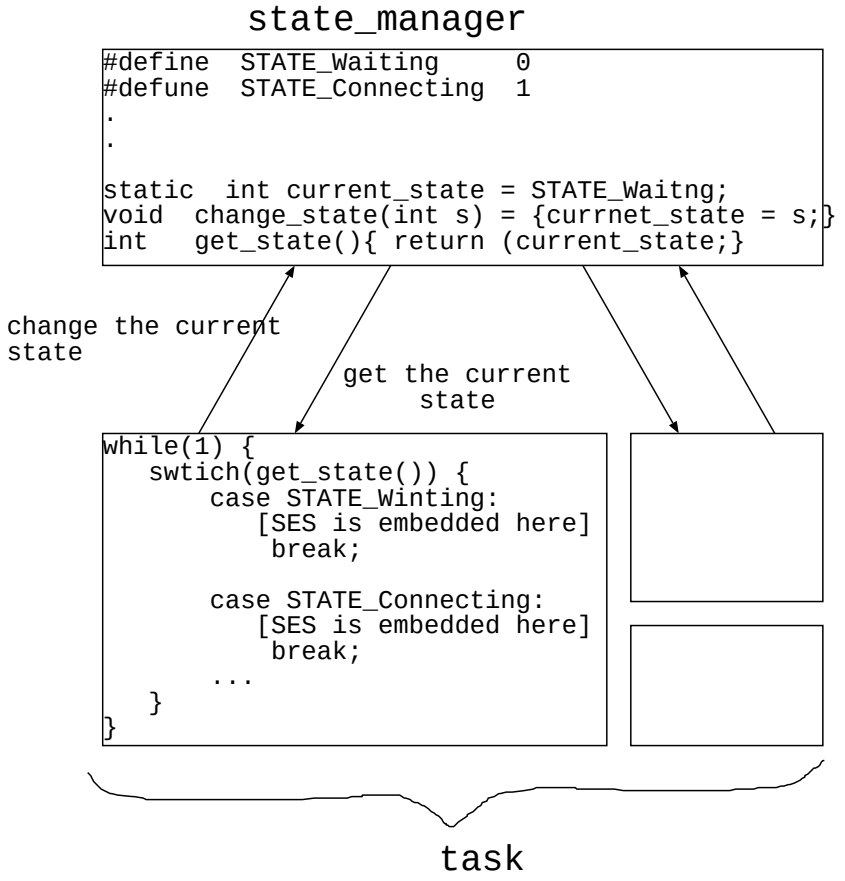


図 2.2: リファレンスアーキテクチャ

て、属性は変数として実装される。

SES コンポーネント

SES モデルに出現する個々の SES は、SES コンポーネントとして実装される。SES コンポーネントは、if 文による制御ブロックと関数呼び出しの列により構成される。SES による分岐は if 文、処理は関数呼び出し、イベントは状態遷移図の現在状態を変更する関数呼び出しとして実装される。

第 3 章

自動生成システム

3.1 自動生成システムの概要

本研究で提案する自動生成システムは、SES モデルと実装に関する情報から、RTOS 上のプログラムを生成するものである。SES モデルは SES と呼ばれる処理列の集合により構成されている。SES はオブジェクトが持つメソッドの呼び出し列であり、設計工程では、SES を用いて対象となるソフトウェアの振舞を記述する。一方、RTOS 上のソフトウェアは、タスクの集合とそれらのスケジューリング法により構成されており、個々のタスクがスケジューリング法に従い、並行に実行される。

そこで提案する自動生成システムは、SES モデルに記述されている SES の集合とを RTOS 上で動作するタスクとスケジューリング法が記述されているプログラムに変換する。

現在提案されている SES モデルには、スケジューリング法やタスクといった実装に関する情報は記述することができない。そこでまず、これらの実装に関する情報を SES モデルに記述できるようにする。

タスクは、どの SES をどのタスクに詰め込むかを指定することで、SES モデルのリファレンスアーキテクチャに基づいて定義することができる。また、SES モデルにおいて SES のスケジューリングが発生する箇所は SES の実行の前か後である。そこで、SES モデルに以下のような拡張を行う。

- SES モデルに出現する SES に詰め込むタスクを割り当てることができるように拡張する。
- SES モデルに出現する SES の前後に RTOS のスケジューリングに関するシステムコールを埋め込むことができるように拡張する。

SES モデルの拡張を行ったことで、自動生成システムの入力を以下のように定めることができた。

- SES モデル
- SES のタスクへの詰め込み
- SES のスケジューリング

3.2 自動生成システムの入力

??節では、SES モデルの拡張を行い、自動生成システムの入力を SES モデル、SES のタスクへの詰め込み、SES のスケジューリングと定めた。ここでは、自動生成システムの各入力について詳しい説明を行う。

自動生成システムへの入力の記述言語に関しては、??章で説明を行う。

3.2.1 SES モデル

SES モデルに関する入力は、SES と状態の関係、イベント、リファレンスアーキテクチャの３つである。

- SES と状態の関係
  SES モデルには SES が複数の状態に存在することができる。このため、プログラムを自動生成するには、付随する状態によって SES を区別する必要がある。そこで、自動生成システムの入力に SES がどの状態に付随しているか、を入力として記述する。

- イベント
  SES モデルに出現するイベントは、SES が状態を遷移するために出力される。このため、同一の SES でも付随している状態が異なれば、イベントによって遷移する先の状態も異なってくる。そこで、本システムでは各状態に関してイベントと遷移先の状態名の組を記述する。SES はどの状態に付随するかは、**SES と状態の関係**の記述から分かっている。以上から、各状態で SES がもつイベントとその遷移先の状態を特定することができるようになり、同一の SES が異なる状態で同じイベントを出力しても各々の遷移先の状態に遷移することができるようになる。

- リファレンスアーキテクチャ
  リファレンスアーキテクチャは以下の３つで構成されている。このため、個々に自動生成システムへの入力を考える。

  – 状態マネージャー
    状態マネージャーの現在状態を表す変数や現在状態を獲得する関数は SES モデルに組み込みの関数である。このため、状態マネージャーを作成するのに必要な情報は SES モデルに出現する状態に関する情報だけである。しかし、SES モデルの出現する状態に関する情報は、既に**SES と状態の関係**の入力で与えている。このため、自動生成システムの状態マネージャーに関する入力は特に必要がない。

  – SES コンポーネント
    SES コンポーネントは個々の SES を実装したものであり、分岐、処理、イベントで構成される。SES コンポーネントには処理のなかに、時間に関する関数呼び出しを行うものがある。そこで本研究では、時間に関する関数呼び出しはスケジューリングに強い影響を与えるため、処理とは区別し**タイマー**として考える。このため。SES コンポーネントの入力は、分岐、処理、タイマー、イベントの４つの項目にわけて記述する。

  – 機能コンポーネント
    機能コンポーネントは SES モデルに出現するオブジェクトを実装したもので、C 言語の変数と関数により構成されている。自動生成システムへの入力はそのまま C 言語の関数として与えることにする。

3.2.2 SES のタスクの詰め込み

RTOS 上で動作するプログラムを生成するには、SES をタスクに詰め込むことが必要である。ここでは、SES をどのタスクに詰め込んだのかにという事について記述を行う。

3.2.3 SES のスケジューリング

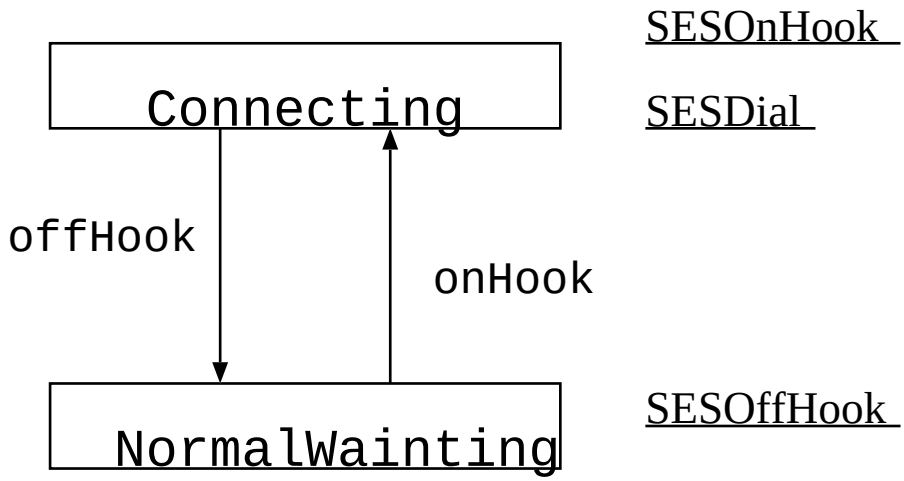
SES のスケジューリングは SES の実行の前後に発生する。このため、RTOS のスケジューリングに関するシステムコールを SES の前後に埋め込むことでスケジューリングに関す

る記述を行う。

3.3 入力の記述言語

3.3.1 SES モデルに関する記述言語

図??の SES モデルを例にとり SES モデルに関する記述言語について説明を行う。



例題の入力は以下のようになる。

自動生成システムの入力

```
state  Waiting   |  [ SESOnHook. SESDial ]
state  Conecting |  [ SESOffHook ]
```

ここで、Waiting, Conecting は状態名、SESOffHook、SESDial、SESOffHook は、SES の名前である。状態 Waiting には、SESOffHook、SESDial が、状態 Conecting には、SESOffHook が付随していること表す。

イベント

SES モデルに出現するイベントも記述する。ここでは、識別子**trans** を用い、次にイベントを出力する SES が存在している状態名を記述する。<trans_list>の箇所には、イベント名とそのイベントが遷移する先の状態名を記述する。

自動生成システムの文法

```
<statement> ::= trans <状態名> '|' "[" <trans_list> "]"

<trans_list> ::= '[' <イベント名> '.' <遷移先の状態名> ']'
                                                    <trans_list>
              |
              ;
```

例題の入力は以下のようになる。

自動生成システムの入力

```
trans Waiting   | [ [offHook. Conecting ] ]
trans Conecting | [ [onHook, Waiting ] ]
```

ここで、イベント**offHook** が、状態 Waiting から状態 Conecting に遷移するイベントであることを表現している。同様に、イベント**onHook** が、状態 Conecting から、状態 Waiting に遷移するイベントであることも表現している。

状態マネージャー

状態マネージャーを構成するための情報 (SES モデルに出現する状態名) は、既に入力として与えているので、自動的に作成する。なお、提案する自動生成システムでは、初期状態は自動的に決定されてしまう。(以下の例では、初期状態が自動的に Waiting になっている) このため、初期状態はプログラム生成後に希望する初期状態を記述しなければならない。

- 例題の状態マネージャーの出力結果

状態マネージャー

```
#define Waiting     0
#define Conecting   1
.
.

static int current_state = Waiting; /* 初期状態 */

void change_state(int s){ current_state = s; }
int get_state(){ return ( current_state); }
```

SES コンポーネント

SES コンポーネントは識別子**comp** を用いて記述する。SES コンポーネントを分岐、処理、タイマー、イベントの４つに分類したので、分岐には**branch** を、処理には**process** を、タイマーには**timer** を、イベントには**event** をそれぞれの識別子として定義した。

branch に続いて記述する<SES コンポーネント名>には、条件分岐を行う関数を記述する。

条件分岐を評価した結果が、真であれば最初の<elem>が実行され、偽であれば２つめの<elem>が実行される。識別子**event** の後にはイベント名を記述するが、イベントの入力から得られた情報に基づき、change_state(**遷移先の状態名**) の形で出力される。change_state() は現在状態を変更する SES モデルの組み込みの関数である。

自動生成システムの文法

```
<statement> ::= comp <SES の名前> '|' "[" <elem> "]"

<elem> ::= branch  <SES コンポーネント名> '?'
                        '[' <真の場合に実行される elem> ']'
                        '[' <偽の場合に実行される elem> ']'
         |  process <SES コンポーネント名>  '.' <elem>
         |  time <SES コンポーネント名>
                        [ <ID 番号> '.'
                          <起動周期期間>'.'
                          <実行させる SES 名>
                        ] '.' <elem>
        |  event   <イベント名>
        |
        ;
```

図??の SESOnHook の SES コンポーネントの記述言語は以下のようになる。

自動生成システムの入力

```
comp SESOnHook | [ bra HookGetState ?
[]
[pro ReceiveMikeOff.
 pro ReceiveSpeakOff.
 pro DialPanelTurnOff.
 pro LineClose.
 eve onHook]
]
```

まず分岐 HookGetState() で条件分岐をおこない、真ならば、何もせず、他の SES に実行制御を渡すことになる。偽ならば、ReceiveSpeakOff() 以下の処理を順に実行し、最後にイベント onHook を出力し、イベントを出力し状態遷移を行う。

次に、SESRecorderOn を例に挙げて識別子 time で定義された入力がどのような形で出力されることになるかを説明する。

自動生成システムの入力

```
comp SESRecorderOn        | [
   pro AutoRecorderStartMessage.
   pro AutoRecorderStartRecording.
   time startTimer2 [2. 60. SESRecorderTimeOut].
   eve startRecording]
```

識別子 time で定義されている startTimer2 は、60 秒後に SESRecorderTimeOut を実行させる関数である。

本研究ではこのような実時間制約があるような SES は周期ハンドラを用いて実装する。周期ハンドラは、周期ハンドラを定義するシステムコール def_cyc、周期ハンドラを活性化させるシステムコール act_cyc() と周期ハンドラによってタスクを起動するシステムコール (wup_tsk()) を用いて実現する。

まず、startTimer2 関数を以下のように出力する。

自動生成システムの出力

```
void
startTimer2()
{
        act_cyc(CYC2_ID, TCY_ID | TCY_INI);
}
```

act_cyc() システムコールは、周期ハンドラの活性状態を変更するシステムコールである。CYC2_ID は周期ハンドラ指定番号を示し、TCY_ID | TCY_INI は活性状態を ON にするのと同時に、周期カウントをクリアする。これにより、act_cyc を実行してからちょうど指定周期が経過した後に、最初の周期ハンドラが実行される。

また、周期ハンドラは以下の関数によって定義することができる。以下の関数で、cycid は周期ハンドラ ID 番号、cyc_time は指定周期を表している。周期ハンドラの ID 番号は define 文で定義され整数が割り当てられる。また、timer_Handle2 は周期ハンドラで制御するタスクを指定する関数である。

自動生成システムの出力

```
ER
create_start_cych2(ID cycid, CYCTIME cyc_time)
{
        T_DCYC   pk_dcyc;
        ER ercd;
        pk_dcyc.exinf = 0;
        pk_dcyc.cycatr = TA_HLNG;
        pk_dcyc.cychdr = (FP) Timer_Handler2;
        pk_dcyc.cycact = TCY_ON;
        pk_dcyc.cyctim = cyc_time;

        ercd = def_cyc(cycid, &pk_dcyc);

        if(ercd != E_OK){
           syslog(LOG_NOTICE, "error %d with def_cyc(%d)",
                                        ercd, cycid);
        }
        return(ercd);
}
```

次に周期ハンドラを実行するタスクを決定する。仮に、SESRecorderTimeOut が**task5**に詰め込まれているとすると、timer_Handle2 関数は以下のようになる。

自動生成システムの出力

```
Timer_Handler1()
{
        wup_tsk(task5);
}
```

以上、３つの関数を生成することで周期ハンドラの実現が可能となる。

機能コンポーネント

機能コンポーネントは、C 言語の関数と変数により構成される。オブジェクトが持つ処理は関数として、属性は変数として実装する。機能コンポーネントは、識別子**func** に続いて関数名を記述し、そのまま C 言語の形で記述する。

自動生成システムの文法

```
<statement> ::= func '|' <機能コンポーネント名> <body_list>

<body_list> ::=
              |  <文字ブロック> <body_list>
              ;
```

ここでは、上記で例にあげたSESOnHook の機能コンポーネントの一部(`HookGetState()` と `ReceiveMikeOff()`) を例に挙げて示す。

自動生成システムの入力

```
func |
HookGetState()
{
if (hook_state == 1) {
return (1);
}
else {
return (0);
}
}

func |
ReceiveMikeOff()
{
if (r_mike_state == 1) {
printf("SESOnHook ReceiveMikeOff\n\r");
r_mike_state = 0; /* OFF にもどす  */
}
}

func |
static int hook_state = 0;
static int r_mike_state = 0;
```

ここで、HookGetState() 関数は、受話器が電話においてあるかどうか、を確認する関数である。この関数は、受話器がおいてあるなら 0を受話器がはずれているなら 1 を返す。

変数 hook_state はこの受話器の状態を表す (0であれば受話器は置いてあり、1であれば受話器は外れていることを表す) 大域変数である。ReceiveMikeOff() 関数は、レシーバーマイクを OFF にする関数である。r_mike_state 変数は、レシーバーマイクの状態

を表す大域変数であり、１であればマイクON、０であればOFFであることを表す。この r_mike_state 変数の値が１であれば (つまりマイクがON)、変数 r_mike_state の値を０ (つまりレシーバーマイクをOFF) にする。

上記で説明したような状態を表す大域変数 (hook_state,r_mike_state など) は機能コンポーネントの入力の最後に記述する。

3.3.2 SESのタスクへの詰め込みに関する記述言語

RTOS上で動作させるためには、SESモデルに存在するSESをタスクに詰め込まなければならない。そこで、どのようなSESのタスクに詰め込み方を行ったかについて記述する。??ページの図??を例にすると、以下のような記述になる。

- 入力例

自動生成システムの入力

```
task 1 | [SESOnHook. SESOffHook ]
task 2 | [SESDial]
```

この入力では、タスクを２つ定義している。タスク１には、SESOnHookとSESOffHookを詰め込み、タスク２には、SESDialを詰め込むことを記述している。

3.3.3 SESのスケジューリングに関する記述言語

SESのスケジューリングはSESの前後にシステムコールを埋め込むことで行う。このため、SESのスケジューリングに関する記述はSESの前に置くシステムコールと後に置くシステムコールを入力として記述する。そこで、SESのスケジューリングに関する記述の言語を定義した。

??ページの図??を例にすると、以下のような入力になる。

自動生成システムの入力

```
ses SESOffHook    | [][tslp_tsk(500)]
ses SESOnHook     | [][tslp_tsk(500)]
ses SESDial       | [][tslp_tsk(100)]
```

上記の定義では、SESOffHook、SESOnHook を 500ms 毎に活性化、SESDial を 100ms 毎に活性化するようなスケジューリングを定義している。ここで、`tslp_tsk()` は、自タスクを起床待ちにする**ItIs** のシステムコールである。

3.3.4 記述言語

自動生成システムの全ての入力の記述言語を以下に示す。

自動生成器の入力の記述言語

```
<program> ::= <statement> <program>
            |  <statement>

<statement> ::= TASK  <name>  '|'  '[' <ses_list> ']'
            |  STATE <name>  '|'  '[' <ses_list> ']'
            |  SES   <name>  '|'
                  '[' <before_block> ']''['<after_block> ']'
            |  COMP  <name>  '|'  '[' <elem> ']'
            |  TRANS <name>  '|'  '[' <trans_list> ']'
            |  FUNC   '|'  <name>  <body_list>
            ;

<body_list>  ::=  <name> <body_list>
              ;

<trans_list>  ::=  '[' <name> '.' <name> ']' <trans_list>
              |
              ;

<elem>    ::=  BRANCH   <name> '?'
                    '[' <elem> ']' '[' <elem> ']'
           |   PROCESS  <name> '.' <elem>
           |   TIME     <name>
                 '[' <name> '.' <name> '.' <name> ']' '.' <elem>
           |   EVENT    <name>
           |
           ;

<ses_list> ::= <name> '.' <ses_list>
            |  <name>
           ;

<before_block> ::= <name>
<after_block>  ::= <name>

<name> ::= [0-9a-zA-Z_(){}=;%\\*,"\-\+\>\#]+
```

3.4 自動生成の手法

この節では、定義した入力の記述から RTOS 上で動作するプログラムを自動生成するアルゴリズムについて説明する。

まず、SES モデルには RTOS 上で動作させるためのリファレンスアーキテクチャがあることについては??章で説明した。本研究で提案する自動生成システムはこのリファレンスアーキテクチャにタスク名や SES を埋め込むことで RTOS 上で動作するプログラムを生成する。図??はリファレンスアーキテクチャのタスクの箇所を示したものである。こ

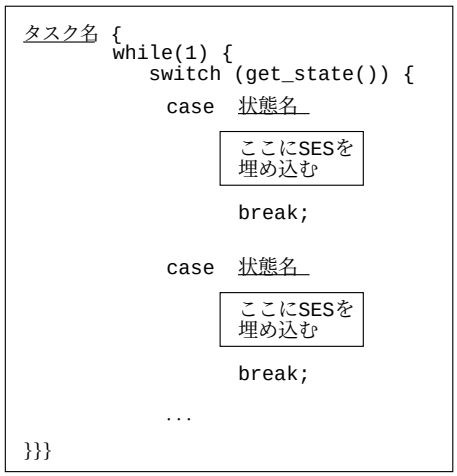


図 3.2: SES モデルのテンプレート

の図にはタスク名と状態名と SES を埋め込む場所があらかじめ定義されている。そこで、図??に順次、タスク名と状態名と SES を埋め込んでいくことで自動生成を行う。

自動生成する手順について説明する。

- 1

  SES のタスクの詰め込みに関する記述を基に、タスクに付随する SES をまとめた表を作成する。

task1	task2
SESOffHook	SESDail
SESOnHook	

表 3.1: タスクと SES

- 2

  表??と SES モデルの状態に関する情報から、タスクと状態に関する図??のような表 (中間コード) を作成する。

	task1	task2
NormalWaiting	SESOffHook	
Connecting	SESOnHook	SESDail

表 3.2: 中間コード

- 3

  (2) で作成した表を基に、テンプレートにタスク名、状態名、SES、スケジューリングに関するシステムコールを順次、埋め込んでいく。

```
task1(INT tskno){
while(1) {
switch(get_state()) {

case NormalWaiting:

        (システムコール)
        [SESOffHook の SES コンポーネントを埋め込む]
        (システムコール)

break;

case Connecting:

        (システムコール)
        [SESOnHook の SES コンポーネントを埋め込む]
        (システムコール)

break;
}}}
```

- 4

  最後に、状態と SES の関係を記述した入力から状態マネージャを作成する。

第 4 章

電話機システムの実装

4.1 実装環境

現在、様々な組み込みシステムで、μIRTON 準拠のカーネルが開発され、そして使用されている。本研究では ItIs と呼ばれる μIRTON 準拠のリアルタイムカーネルを用いる。

4.2 ItIs

4.2.1 タスク管理機構

μIRTON では、並行処理されるプログラム単位をタスクと呼ぶ。タスクを管理するためには、タスクの ID、タスクの状態、タスクの優先度などがある。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する。

タスクの状態

タスク状態には、実行状態 (RUN)、実行可能状態 (READY)、待ち状態 (WAIT)、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-EXEISTENT) がある。

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、実行可能状態にあるタスクのなかで、最も高い優先度を持つタスクが実行状態にある。実行可能状態と実行状態にあるタスクは、タスクの優先度の順に、列 (レディキュー) を作り、その列の先頭のタスクから順に実行状態になる。

4.3 電話機システムの概要

本研究で例題に挙げた電話器のシステムには以下のような機能がある。

- 着信処理機能
  着信処理とは、他の電話器からの接続要求を受けたときに、応答して電話器の着信音を鳴らす機能である。

- 発信機能
  発信機能は、相手先への接続を要求する機能である。

- 留守番機能
  他の電話器からの接続要求があったとき、一定時間着信音を鳴らした後に、自動的に回線を繋いで相手からのメッセージの録音を開始する機能である。

4.4 例題の電話機の入力

前章の自動生成システムの入力に関する説明に従い、SES モデル、SES のタスクへの詰め込み、SES のスケジューリングに関する記述を行う。

4.4.1 SES モデルに関する記述言語

電話機の SES モデル

SES と状態の関係

まず、図??から SES モデルの SES と状態の関係についての入力を記述する。自動生成機への入力を以下に示す。

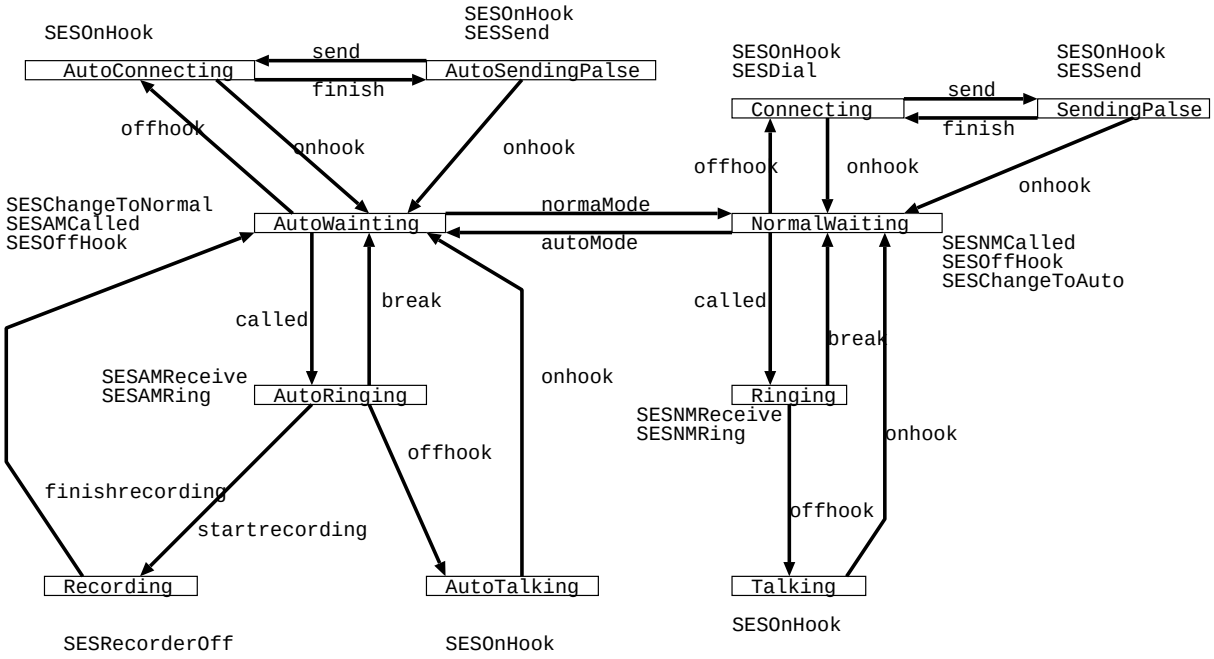


図 4.1: 電話機の SES モデル

自動生成器の入力

```
state NormalWaiting    | [SESOffHook . SESNMCall. SESChangeToAuto]
state Connecting       | [SESOnHook. SESDial]
state SendingPulse     | [SESOnHook. SESSend]
state Ringing          | [SESNMRing. SESNMReceive]
state Talking          | [SESOnHook]
state AutoWaiting      | [SESChangeToNormal. SESAMCall. SESOffHook]
state AutoConnecting   | [SESOnHook. SESDial]
state AutoSendingPalse | [SESOnHook. SESSend]
state AutoRinging      | [SESAMReceive. SESAMRing. SESRecorderOn]
state Recording        | [SESRecorderTimeOut. SESRecorderOff]
state AutoTalking      | [SESOnHook]
```

イベント

今回の電話機の例題のSESモデルでは、イベントは、offHook, onHook, send, finish, called, br
,finishRecording, startRecording, autoMode, normalMode の10個ある。以下で各状態についてイベントの入力記述を行う。自動生成機への入力を以下に示す。

- NormalWaiting
  イベント autoMode はオートモードへのモード変換をしたとき、イベント offHook はオフフックしたとき、イベント called は呼び出し(コール)がおきたとき、に各々のSESによって出力される。

自動生成システムの入力

```
trans NormalWaiting    | [
     [ offHook. Connecting ]
     [ called. Ringing]
     [ autoMode. AutoWaiting ]
]
```

- Conecting
  イベント send はパルス送信時に、イベント onHook はオンフックしたときに、各々のSESによって出力される。

自動生成システムの入力

```
trans Connecting       | [
     [ onHook.NormalWaiting ]
     [ send. SendingPalse ]
]
```

- SendingPalse
  イベント finish はパルス送信終了時に、イベント onHook は、会話終了時にオンフックしたときに、各々のSESによって出力される。

自動生成システムの入力

```
trans SendingPalse          | [
     [ onHook. NormalWaiting ]
     [ finish. Connecting ]
]
```

- Ringing

  イベント break は着信音 (コール) が終了したときに、イベント offHook は、電話に出るためにオフフックしたときに、各々の SES によって出力される。

自動生成システムの入力

```
trans Ringing          | [
     [ break. NormalWaiting]
     [ offHook. Talking]
]
```

- Talking

  イベント onHook はオンフックしたときに SES によって出力される。

自動生成システムの入力

```
trans Talking          | [
     [ onHook. NormalWaiting]
]
```

- AutoWaiting

  イベント normalMode はノーマルモードへのモード変換したとき、イベント offHook はオフフックしたとき、イベント called は呼び出し (コール) がおきたとき、に各々の SES によって出力される。

自動生成システムの入力

```
trans AutoWaiting      | [
     [ offHook. AutoConnecting]
     [ called. AutoRinging]
     [ normalMode. NormalWaiting]
]
```

- AutoConnecting

  イベント send はパルス送信時に、イベント onHook はオンフックしたときに、各々の SES によって出力される。

自動生成システムの入力

```
trans AutoConnecting   | [
     [ onHook. AutoWaiting]
     [ send. AutoSendingPalse]
]
```

- AutoSendingPalse

  イベント finish はパルス送信終了時に、イベント onHook はオンフックしたときに、各々の SES によって出力される。

自動生成システムの入力

```
trans AutoSendingPalse | [
     [ finish. AutoConnecting]
     [ onHook. AutoWaiting]
]
```

- AutoRinging

  イベント break は着信音 (コール) が終了したときに、イベント offHook は、電話に出るためにオフフックしたときに、イベント startRecording は、メッセージ録音を開始したときに、各々の SES によって出力される。

自動生成システムの入力

```
trans AutoRinging      | [
     [ break. AutoWaiting]
     [ startRecording. Recording]
     [ offHook. AutoTalking]
]
```

- AutoTalking

  イベント onHook はオンフックしたときに SES によって出力される。

自動生成システムの入力

```
trans AutoTalking      | [
     [ onHook. AutoWaiting]
]
```

- Recording

  イベント finishRecording はメッセージ録音終了時に、SES によって出力される。

自動生成システムの入力

```
trans Recording        | [
     [ finishRecording. AutoWaiting]
]
```

電話機の SES モデルに出現するオブジェクト

次に、SES モデルに出現するオブジェクトについて説明する。

- Panel
  このオブジェクトは今回例題とした電話機のメインパネルを表す。
- DialPanel
  このオブジェクトはダイアルボタンを持ったパネルを表す。
- ReceiveMic
  このオブジェクトは受話器のマイクを表す。
- ReceiveSpeaker
  このオブジェクトは受話器のスピーカを表す。
- Line
  このオブジェクトは電話回線のインターフェイスを表す。
- Hook
  このオブジェクトは受話器のフックを表す。受話器を取るとオフフック、受話器を置くとフックオンとなる。
- Bell
  このオブジェクトは着信音を発生させるベルを表す。
- Speaker
  このオブジェクトはベルの音を出力するスピーカを表す。
- AutoRecorder
  このオブジェクトは、この電話機の自動録音装置を表す。

これらのオブジェクトは RTOS 上で動作するプログラム中では大域変数として表し、自動生成機の入力ではこれらの変数をまとめて１つの機能コンポーネントとし、機能コンポーネントの入力記述の最後に記述する。

自動生成システムの入力

```
func |
static int panel_state = 0;
static int dial_lamp_state = 0;
static int receiver_mic_state = 0;
static int receiver_speak_state = 0;
static int line_state = 0;
static int hook_state  = 0;
static int palse_state = 0;
static int start_detect_state = 0;
static int call_state = 0;
static int speak_state = 0;
static int record_state = 0;
```

SES コンポーネント

以下で、SES モデルに出現する全ての SES について説明し、その後に自動生成機への入力についても述べる。

- SESOffHook
  1. 受話器フックがフックオフを検出する。
  2. 受話器マイクのスイッチを ON にする。
  3. 受話器スピーカのスイッチを ON にする。
  4. ダイアルパネルのランプを点灯させる。
  5. 電話回線をオープンする。
  6. ダイアルパネルからの電話番号の入力検知を開始する。
  7. イベント offHook を出力する。

自動生成システムの入力

```
comp SESOffHook |  [ bra HookGetState ?
[pro ReceiveMikeOn.
 pro ReceiveSpeakOn.
 pro DialPanelTurnOn.
 pro LineOpen.
 pro DialPanelStartDetect.
 eve offHook]
[]
]
```

- SESDial
  1. ダイアルパネルに電話番号を入力する。
  2. 電話回線にパルスを送信する。
  3. イベント send を出力する。

自動生成システムの入力

```
comp SESDial | [ bra DialPanelDetectButton ?
[pro LineStartSendPalse.
 eve send]
[]
]
```

- SESSend
  1. パルスの送信終了を検知する。

2. ダイアルパネルからの電話番号の入力検知を開始する。
3. イベント finish を出力する。

自動生成システムの入力

```
comp SESSend | [ bra LineDetectEndPalse ?
[pro DialPanelStartDetect.
 eve finish]
[]
]
```

- SESOnHook
  1. 受話器フックがオンフックを検出する。
  2. 受話器マイクのスイッチを OFF にする。
  3. 受話器スピーカのスイッチを OFF にする。
  4. ダイアルパネルの全てのランプを消灯させる。
  5. 電話回線をクローズする。
  6. イベント onHook を出力する。

自動生成システムの入力

```
comp SESOnHook | [ bra HookGetState ? []
[pro ReceiveMikeOff.
 pro ReceiveSpeakOff.
 pro DialPanelTurnOff.
 pro LineClose.
 eve onHook]
]
```

- SESChangeToAuto
  1. モード変換ボタンが入力を検出する。
  2. 自動メッセージ録音器のスイッチを ON にする。
  3. イベント autoMode を出力する。

自動生成システムの入力

```
comp SESChangeToAuto     | [bra  PanelDetectRecordButton ?
[pro  AutoRecordOn.
 eve  autoMode]
[]
]
```

- SESNMCall
  1. 電話機がコールを検出する。
  2. 呼び出し音を鳴らすスピーカのスイッチを ON にする。
  3. 電話機のパネルを点灯させる。
  4. イベント called を出力する。

自動生成システムの入力

```
comp SESNMCall | [ bra LineDetectCall ?
[pro SpeakOn.
 pro PanelLampOn.
 eve called]
[]
]
```

- SESNMReceive
  1. 受話器フックがオフフックを検出する。
  2. スピーカのスイッチを OFF にする。
  3. 受話器スピーカのスイッチを ON にする。
  4. 受話器マイクのスイッチを ON にする。
  5. 電話回線をオープンする。
  6. イベント offhook を出力する。

自動生成システムの入力

```
comp SESNMReceive | [ bra HookGetState ?
[pro SpeakOff.
 pro ReceiveSpeakOn.
 pro ReceiveMikeOn.
 pro LineOpen.
 eve offHook]
[]
]
```

- SESNMRing
  1. 電話機がコールを検出する。
  2. 呼び出し音を鳴らす。

  1. 電話機がコール終了を検出する。
  2. スピーカのスイッチを OFF にする。
  3. ダイアルパネルのランプを消灯する。
  4. イベント break を出力する

自動生成システムの入力

```
comp SESNMRing | [ bra LineDetectCall ?
[pro  BellRing]
[pro  SpeakOff.
 pro  DialPanelTurnOff.
 eve  break]
]
```

- SESChangeToNormal
  1. モード変換ボタンが入力を検出する。
  2. 自動メッセージ録音器のスイッチを OFF にする。
  3. イベント normalMode を出力する。

自動生成システムの入力

```
comp SESChangeToNormal   | [bra  PanelDetectRecordButton ?
[]
[pro  AutoRecordOff.
 eve  normalMode]
]
```

- SESAMCall
  1. 電話機がコールを検出する。
  2. ダイアルパネルの全てのランプを点灯する。
  3. １０秒後にSESRecorderOnを実行する。
  4. イベントcalledを出力する。

自動生成システムの入力

```
comp SESAMCall           | [bra LineDetectCall ?
[pro   DialPanelLampOn.
 time  startTimer1 [1. 1000. SESRecorderOn].
 eve   called]
[]
]
```

- SESRecorderOn
  1. メッセージの録音開始を知らせるアナウンスを行う。
  2. メッセージの録音を開始する。
  3. ６０秒後にSESRecorderTimeOutを実行する。
  4. イベントstartRecordingを出力する。

自動生成システムの入力

```
comp SESRecorderOn       |
[pro AutoRecorderStartMessage.
 pro AutoRecorderStartRecording.
 time startTimer2 [2. 6000. SESRecorderTimeOut].
 eve startRecording]
```

- SESRecorderTimeOut
  1. メッセージの録音を終了する。
  2. 終了時間を記録する。
  3. ダイアルパネルのランプを消灯する。
  4. イベント finishRecording を出力する。

自動生成システムの入力

```
comp SESRecorderTimeOut  |
[pro  AutoRecorderStopRecording.
 pro  AutoRecorderTimeStamp.
 pro  DialPanelLampOff.
 eve  finishRecording]
```

- SESAMReceive
  1. 受話器フックがオフフックを検出する。
  2. SESRecorderOn の実行を終了する。
  3. 受話器スピーカのスイッチを ON にする。
  4. 受話器マイクのスイッチを ON にする。
  5. 電話回線をオープンする。
  6. イベント offhook を出力する。

自動生成システムの入力

```
comp SESAMReceive        | [bra  HookGetState ?
[time  stopTimer1 [1].
 pro   ReceiveSpeakOn.
 pro   ReceiveMikeOn.
 pro   LineOpen.
 eve   offHook]
[]
]
```

- SESAMRing
  1. 電話機がコール終了を検出する。

2. SESRecorderOn の実行を終了する。

3. イベント break を出力する。

自動生成システムの入力

```
comp SESAMRing            | [bra  LineDetectCalled ? []
[time  stopTimer1 [1].
 eve   break]
]
```

- SESRecorderOff

1. 受話器フックがオンフックを検出する。
2. SESRecorderTimeOut の実行を終了する。
3. メッセージの録音を終了する。
4. 終了時間を記録する。
5. ダイアルパネルのランプを消灯する。
6. イベント finishRecording を出力する。

自動生成システムの入力

```
comp SESRecorderOff       | [ bra  LineIsConnect ?
[]
[time  stopTimer2 [2].
 pro   AutoRecorderStopRecording.
 pro   AutoRecorderTimeStamp.
 pro   DialPanelLampOff.
 eve   finishRecording]
]
```

機能コンポーネント

機能コンポーネントは SES モデルに出現するオブジェクトの実装であることは、以前に述べた。この節では、自動生成システムへの機能コンポーネントの入力について説明する。

ここでは、SESRecorderOn を例に挙げて説明を行う。SESRecorderOn の機能コンポー

ネントとして以下のものがある。

- AutoRecorderStartMessage
  録音開始を示唆するメッセージを出力する関数。

  自動生成システムの入力

```
func |
AutoRecorderStartMessage()
{
printf("AutoRecorderStartMessage\n\r");
fflush(stdout);
}
```

- AutoRecorderStartRecording
  伝言を録音を開始する関数。

  自動生成システムの入力

```
func |
AutoRecorderStartMessage()
{
printf("AutoRecorderStartMessage\n\r");
fflush(stdout);
}
```

- startTimer2 [2. 40. SESRecorderTimeOut]
  60 秒後にタイムアウトを行う SESRecorderTimeOut を実行させる関数。

  自動生成システムの入力

  なし。

  周期ハンドラに関して定義された関数は出力の形があらかじめ分かっているので関数の中身を自動生成して出力するので入力は不要である。

- startRecording
  状態 `Recording` への状態を遷移を行う関数。

自動生成システムの入力

なし。

SES モデルのイベントは change_State(遷移先の状態名) の形で自動生成される。関数 change_State() は状態マネージャーで定義されている組み込みの関数である。

4.5 SES のタスクへの詰め込みに関する記述言語

SES のタスクの詰め込み方法は、今回は以下のように定義した。

以下で示す自動生成機の入力において重複して SES が記述されているのは、同一タスクでも異なる状態に同じ SES を埋め込むことが可能である。このため、自動生成するには各状態に埋め込む SES を全て記述することが必要にとなる。

- task1

  task1 には、大域変数 hook_state で制御する SESOnHook と SESOffHook、SESAMReceive、SESNMReceive を詰め込む。

自動生成システムの入力

```
task 1 | [
  SESOnHook.
  SESAMReceive.
  SESOnHook.
  SESOnHook.
  SESOffHook.
  SESNMReceive.
  SESOnHook.
  SESOnHook.
  SESOnHook.
  SESOffHook
]
```

- task2

  task2 には、大域変数 call_state で制御する SESNMCall、SESAMCall、SESNM-

Ring、SESAMRing、SESRecorderOff をそして残りの状態に SESSend、SESDial を詰め込む。

自動生成システムの入力

```
task 2 | [
  SESRecorderOff.
  SESAMRing.
  SESDial.
  SESSend.
  SESAMCall.
  SESNMRing.
  SESSend.
  SESSend.
  SESDial.
  SESNMCall
]
```

- task3

  task3 には、大域変数 record_buttun_state(この変数は現在のモードを表す) で制御する SESChangeToAuto、SESChangeToNormal を詰め込む。

自動生成システムの入力

```
task 3 | [
  SESChangeToNormal.
  SESChangeToAuto
]
```

- task4

  task4 には、SESRecorderOn を詰め込む。

自動生成システムの入力

```
task 4 | [
  SESRecorderOn
]
```

- task5

  task5 には、SESRecorderTimeOut を埋め込む。

自動生成システムの入力

```
task 5 | [
  SESRecorderTimeOut
]
```

task4、task5 にSES をそれぞれ１個ずつ独立のタスクとして詰め込んだのは、SESRecorderOn、SESRecorderTimeOut には実時間制約があり、これらを周期ハンドラとして実装するためである。この実時間制約や周期ハンドラに関しては??章で詳しく説明する。

4.6 SES のスケジューリングに関する記述言語

SES のスケジューリングに関して実時間制約のあるものとないものとに分けて考えていく。

4.6.1 実時間制約がない SES のスケジューリング

実時間制約のない SES は、今回はすべて `tslp_tsk(100);` システムコールを用いることで行った。これにより１００ ms 毎にタスクが活性化される。このシステムコールはタイムアウトつきの自タスクを起床待ち状態に移行するシステムコールである。

自動生成システムの入力

```
/* SES のスケジューリングの定義 */
ses SESOffHook          | [][tslp_tsk(100)]
ses SESDial             | [][tslp_tsk(100)]
ses SESSend             | [][tslp_tsk(100)]
ses SESOnHook           | [][tslp_tsk(100)]
ses SESNMCall           | [][tslp_tsk(100)]
ses SESNMReceive        | [][tslp_tsk(100)]
ses SESRecorderOff      | [][tslp_tsk(100)]
ses SESAMRing           | [][tslp_tsk(100)]
ses SESAMCall           | [][tslp_tsk(100)]
ses SESAMReceive        | [][tslp_tsk(100)]
ses SESChangeToAuto     | [][tslp_tsk(100)]
ses SESChangeToNormal   | [][tslp_tsk(100)]
```

4.6.2 実時間制約がある SES のスケジューリング

例題に用いた電話機では３つの実時間制約がある。

- 1
  SESAMCall の実行した後、１０秒後に SESRecorderOn を実行する。
- 2
  SESRecorderOn 実行した後、６０秒後に SESRecorderTimeOut を実行する。
- 3
  １秒毎に、SESNMRing を実行する。

これらのタイマーに従って起動するタスクは周期ハンドラを用いて実装する。今回実装を行う ItIs における周期起動ハンドラはタスク独立部として扱われるために通常のタスクよりも優先して実行される。この電話機の例題では以下の２つの周期ハンドラを使用した。

- Timer_Handle1
  着信音発生時から１０秒後にメッセージ録音を開始する周期起動ハンドラ。

- Timer_Handle2
  メッセージ録音開始から６０秒後にタイムアウトして録音を終了する周期起動ハンドラ。

- Timer_Handle3
  １秒毎にSESNMRingを実行する周期起動ハンドラ。これは、１秒毎に着信音(コール)の状況を確認し、コールが続いているならばベルを鳴らし、コールが途絶えたのならベルのスピーカOFF、ダイアルパネルランプを消灯する。

自動生成システムの入力

```
ses SESRecorderOn        | [slp_tsk()][]
ses SESRecorderTimeOut   | [slp_tsk()][]
ses SESNMRing            | [slp_tsk()][]
```

4.7 例題の電話機の出力

以下で示す自動生成システムの出力を初期状態の値を適宜変更し、今回は計算機上でシミレーションを行うためのタスクを１つ増やしてItIs上で動作を行った。

4.7.1 外部環境とのインターフェイス

本研究では、外部環境とのインターフェイスを実現するタスクを実装している。以下で示す振舞を独立したタスクとしてシミュレーション環境の構築を行った。今回実装を行った電話機システムは、仮想的に動作をコンピューター上でシミュレーションを行ったものである例えば、オンフックをキー':'に割り当て':'が押されるとオンフックを実現している。これらのキー入力はItisのシステムコールserial_read()を用いて実現している。

- オンフック
  キー':'をオンフックに対応させている。

- オフフック
  キー'/'をオフフックに対応させている。

- 相手からの接続要求の開始
  キー’;’を接続要求開始に対応させている。このキーを入力することで着信音が発生する。

- 相手からの接続要求の終了
  キー’#’を接続要求終了に対応させている。このキーを入力することで着信音が終了する。

- 自動録音モードと標準モードの変換
  キー’!’をモード変換にしている。最初は標準モードだが、’!’を入力することで自動録音モードに切換えられ、再び’!’を入力することで標準モードに戻る。

外部環境とのインターフェイス

```
key_check()
{
    char buf[1];
    serial_read(0, buf, 1);

    switch(buf[0]) {
    case '/':
            printf("hello\n\r"); fflush(stdout);
            hook_state = 1;
            call_state = 0;
            break;
    case ':':
            printf("goodby\n\r"); fflush(stdout);
            hook_state = 0;
            call_state = 0;
            break;
          .
          .
          .
}
}
```

上記で示したキー入力は同一状態で同一のシステムコールを実行する場合など、キーボードからの入力に対するブロックが発生し、システムの振舞を正しく実現することができない。そこで、本研究では、これらのキー入力を専門に扱うタスクを設けることにした。

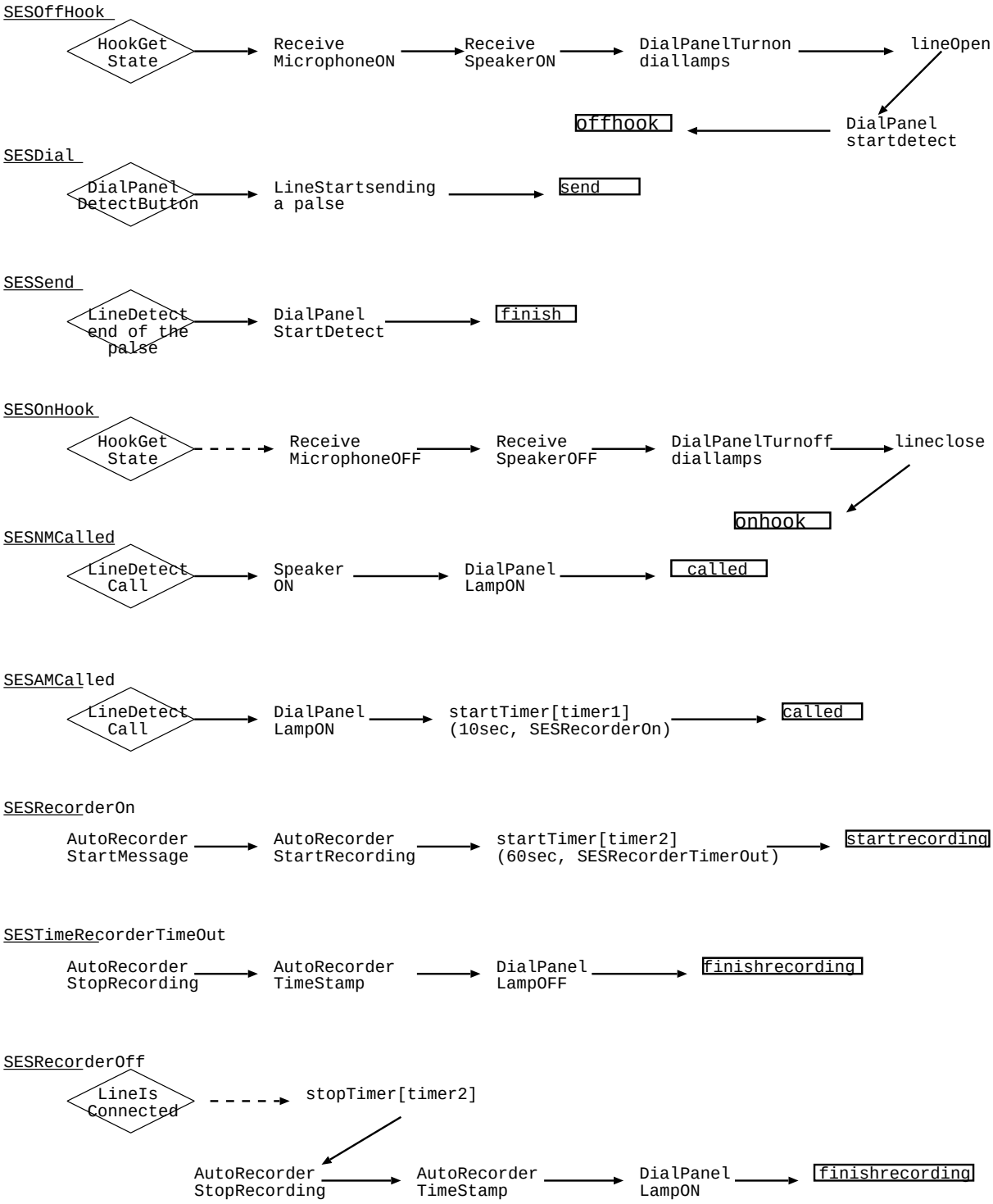

SESOffHook
HookGet State
Receive MicrophoneON
Receive SpeakerON
DialPanelTurnon diallamps
lineOpen
offhook
DialPanel startdetect
SESDial
DialPanel DetectButton
LineStartsending a palse
send
SESSend
LineDetect end of the palse
DialPanel StartDetect
finish
SESOnHook
HookGet State
Receive MicrophoneOFF
Receive SpeakerOFF
DialPanelTurnoff diallamps
lineclose
onhook
SESNMCalled
LineDetect Call
Speaker ON
DialPanel LampON
called
SESAMCalled
LineDetect Call
DialPanel LampON
startTimer[timer1] (10sec, SESRecorderOn)
called
SESRecorderOn
AutoRecorder StartMessage
AutoRecorder StartRecording
startTimer[timer2] (60sec, SESRecorderTimerOut)
startrecording
SESTimeRecorderTimeOut
AutoRecorder StopRecording
AutoRecorder TimeStamp
DialPanel LampOFF
finishrecording
SESRecorderOff
LineIs Connected
stopTimer[timer2]
AutoRecorder StopRecording
AutoRecorder TimeStamp
DialPanel LampON
finishrecording

図 4.2: SES の定義

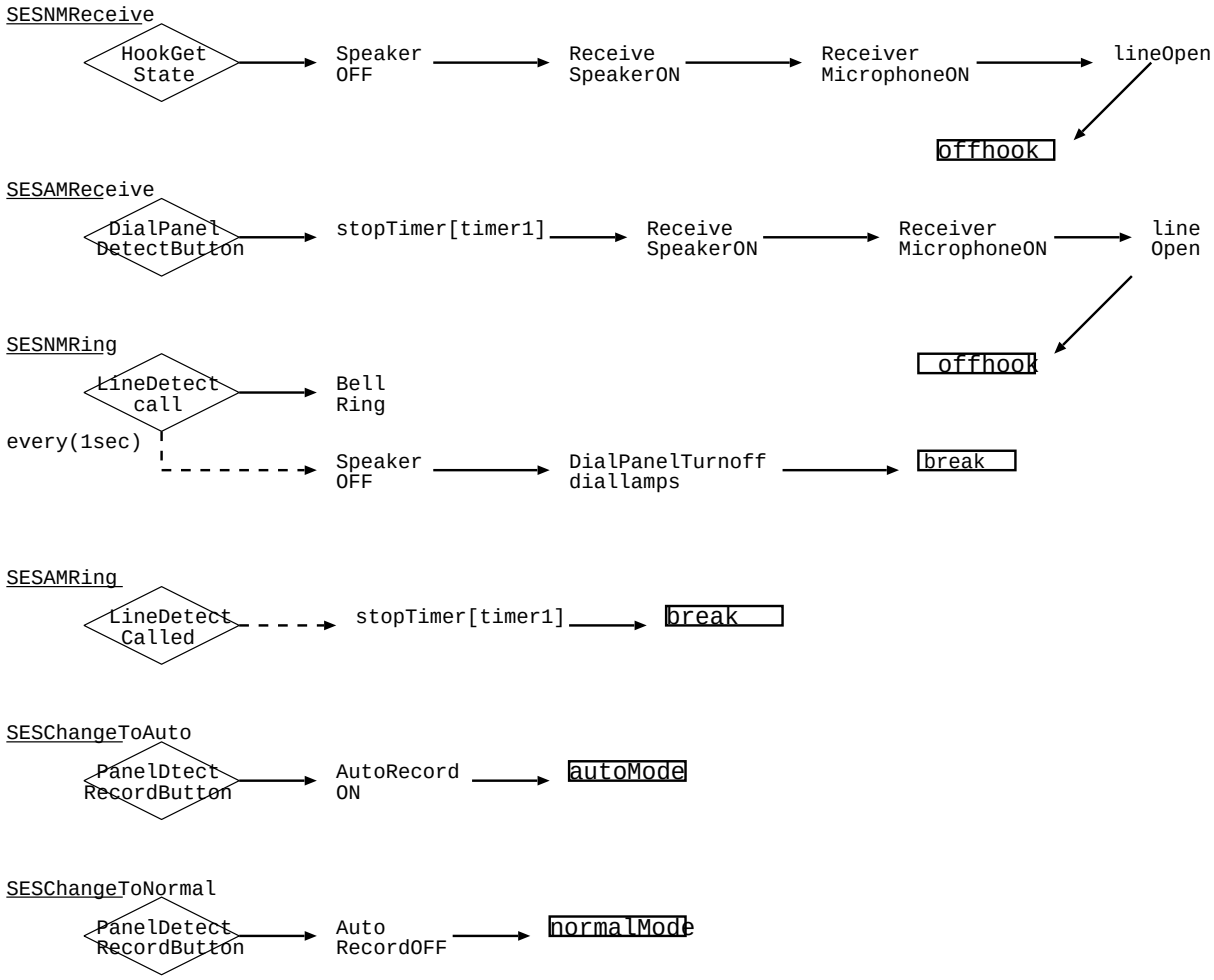

SESNMReceive
HookGet State
Speaker OFF
Receive SpeakerON
Receiver MicrophoneON
lineOpen
offhook
SESAMReceive
DialPanel DetectButton
stopTimer[timer1]
Receive SpeakerON
Receiver MicrophoneON
line Open
offhook
SESNMRing
LineDetect call
Bell Ring
every(1sec)
Speaker OFF
DialPanelTurnoff diallamps
break
SESAMRing
LineDetect Called
stopTimer[timer1]
break
SESChangeToAuto
PanelDtect RecordButton
AutoRecord ON
autoMode
SESChangeToNormal
PanelDetect RecordButton
Auto RecordOFF
normalMode

図 4.3: SES の定義

第 5 章

考察

本研究では、SES モデルから RTOS 上で動作するプログラムを自動生成するシステムの作成を行った。組み込みシステム開発では、実時間制約やハードウェア構成など非機能的要件と呼ばれる厳しい制約があり、分析モデルで抽出したオブジェクトをタスクに割り当てることはできない。このために、組み込みシステムにおいては設計モデルの作成は困難なものとなる。本研究では SES モデルを用いて設計モデルを作成することで、この問題を克服している。

また、設計モデルに SES モデルを採用したことにより、SES モデルの RTOS 上で動作するプログラム作成のためのテンプレートを利用することができる。本研究では SES モデルのテンプレートにいくらかの実装に関する情報を加えることで、RTOS 上で動作するプログラムを自動生成するシステムを作成した。この自動生成システムにより、設計工程と実装工程との対応づけをより明確にしたことで、設計モデルからのプログラムへの変換を容易なものにすることができた。

5.1 自動生成システムの入力について

本研究では、自動生成システムの入力として SES モデル、SES のタスクへの詰め込み、SES のスケジューリングの３つの要件を定義した。本章では、これらの入力の記述をどのように定義したかについての考察を行う。

5.1.1 SES モデル

SES モデルからの入力のうち、SES と状態の関係、イベント、SES コンポーネントは SES モデルから容易に記述することができた。しかし、機能コンポーネントの入力を設計工程から記述するのは困難であった。

機能コンポーネントの変数は、プログラム中に大域変数として表現されるものがほとんどである。このため、プログラム全体を俯瞰しながらでないと、機能コンポーネントの記述は困難であった。そのため、本研究では自動生成システムの機能コンポーネントの入力は、一度、機能コンポーネントを空の関数として出力し、その後にプログラム全体を見ながら記述を行った。しかし、機能コンポーネントの記述を入力として与えることは、ソフトウェアの再利用性の観点から有益であると考える。

5.1.2 SES のタスクへの詰め込み

SES を詰め込むタスクの数は、SES モデルの状態に付随する SES の最大数個は最低限必要である。本研究で提案する自動生成システムでは、SES のタスクへの詰め込みに関する記述には、同一の SES は必ず同一のタスクに詰め込まなければならない、という制約がある。同一の SES を同一のタスクに詰め込むことで、出力プログラム中で同一 SES をタスクのなかにまとめて記述することができ、タスクのスケジューリングを考慮しやすくなる。

5.1.3 スケジューリングについて

本研究で提案した自動生成システムの出力のプログラムは、一部記述を加えなければ動作しない。自動生成を行うアルゴリズムは、テンプレート中の SES を埋め込む場所に順に埋め込んで行くが、状態に埋め込む SES がないときには、SES の埋め込む場所には何も出力されずに以下に示す形で出力される。

自動生成器の出力

```
while(1) {
case NormalWaiting:
        /* 何も出力しない */
break;
.
.
```

このために、無限ループのなかでループを繰り返し続けてしまう。このために、今回の実装では、tslp_tsk(100) システムコールの記述を加えて最終的な実行プログラムとした。これは、タスクに埋め込むSESがない場合でも、スケジューリングに関するシステムコールを自動で出力することでこの問題は回避できる。このシステムコールの自動出力は今後の課題とする。

また、今回のような状況は、SESが記述されていないタスクに状態が遷移していることが原因であり、これはスケジューリング法の変更で対処することができる。

5.2 自動生成システムについて

本研究で提案した自動生成システムは、タスク間通信に関しては実装をしていない。しかし、タスク間通信を除いて今回作成した自動生成システムを電話機の例題に適用し、一部記述を加えた結果、実時間制約を満たすプログラムの生成に成功した。

タスク間通信に関する記述を加えることで、SESモデルからプログラムの完全な自動生成を行うことができるようになる。

第 6 章

まとめ

6.1 まとめ

本研究では、SES モデルから RTOS 上で動作するプログラムを自動生成する自動生成システムの作成を行った。自動生成システムの入力を、SES モデル、SES のタスクへの詰め込み、SES のスケジューリングに定め、自動生成システムへの入力の記述言語を定義した。作成した自動生成システムを電話機を例題として適用し、RTOS 上で動作するプログラムを自動生成することができた。

6.2 今後の課題

SES モデルから完全なプログラムを生成するには、SES モデル、SES のタスクへの詰め込み、SES のスケジューリング、タスク間通信の４つの要件が必要である。

本研究で作成した自動生成システムは、タスク間通信に関する記述は実装していない。今後の課題として本研究で提案した自動生成システムにタスク間通信の記述を加え、さらに様々なスケジューリング法を適用できるような拡張を行う。

謝辞

本研究を行うに当たり, 有益な御指導, 助言および援助を頂きました片山 卓也先生に深く感謝の意を表します. 片山研究室の青木 利晃助手には日頃から有益な御助言, 御支援をいただきました. 感謝いたします. 本論文をまとめるに当たって御協力いただいたソフトウェア基礎講座のメンバーの皆様方に深く感謝致します。

参考文献

[1] T.Aoki and T.Katayama : *SES Model for Object-Oriented Time Critical System Development*, *2000*

[2] 青木利晃 片山卓也: **オブジェクト指向組み込みシステム開発のための設計モデル** *SES* **モデル**,*2000*

[3] 森本大喜: *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**,*1999*

[4] 坂村健: *μITORN3.0* **標準ハンドブック**,*1998*