修 士 論 文

形式仕様記述言語を用いた要求仕様書の形式化と検証仕様書の獲得手法に関する研究

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

吹田有行

2010 年 3 月

修 士 論 文

形式仕様記述言語を用いた要求仕様書の形式化と検証仕様書の獲得手法に関する研究

指導教官　青木利晃 准教授

審査委員主査　青木利晃 准教授
審査委員　島津 明教授
審査委員　二木 厚吉教授

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

710061 吹田有行

提出年月: 2010 年 2 月

概 要

本研究では、仕様書を形式化する為の手法を提案した。仕様書には機能に対してある観点からみた事柄が列挙してあるような機能的記述で書かれている。提案手法により、仕様書から形式化された機能の振る舞いに変換された検証仕様書を獲得した。提案手法を RTOS の１つである OSEK/VDX を用いて問題の検討を行った。

目 次

第1章　はじめに

1.1　背景

車のEUTなどに使用されるリアルタイムオペレーションシステム(以下RTOS)「OSEK/VDX」の仕様書は標準仕様書である。その為、「OSEK/VDX」の仕様書からは様々な実装が作成される。作成した「OSEK/VDX」の実装が仕様書に準拠している事を確認するのは2つの問題点がある為困難である。

1. 仕様書は自然言語で書かれている為曖昧である

2. 仕様書の記述では実装の動作を直接比較することが出来ない

1つ目の問題点はは自然言語で書かれた仕様書の解釈は一意ではない点である大元が曖昧である仕様書から実装が仕様に準拠していることを確認するのは難しい。

2つ目は仕様書の記述は、実装の動作を様々な観点で構成されている点である。

実装は仕様書に記述されたサービスコールによるタスクの状態や資源とタスクの関係や制約が合わさって動作している。しかし、仕様書にはサービスコールによるタスクの状態遷移や制約、資源とタスクの関係や制約が仕様書に分散して別々に記述されている。このため仕様書ではそれぞれの資源やタスク、サービスコールの関係や制約を合わせて考えることは困難である。

よって実装が仕様書に準拠していることを確認するのは困難である。したがって実装が仕様書に準拠しているか確認する為に実装と同じようにサービスコールによるタスクの状態遷移や制約、資源とタスクの関係と制約から実装に対して行う検証の方針を記述した検証仕様書を獲得することが目的である本研究ではこのような仕様書を検証仕様書と呼ぶ。

実装が仕様書に準拠していることを確認する為に検証仕様書の獲得することが求められている。

1.2　検証仕様書

検証仕様書は以下の特徴を持った仕様書である。

1. 実装に対して行われる検証方法が記述されている

検証方法は仕様書から考えだされる。

しかし、OSEK/VDX の標準仕様書は自然言語で書かれている為、仕様書に書かれている制約が抜き出しにくい。その為仕様書を形式化する必要がある。形式化することによって自然言語では行間に埋もれていた制約が明確になる。次に仕様書を形式化する方法を考える。

1.2.1 検証仕様書の作成方法

1.2 章で述べたように仕様書を形式化して、行間に埋もれていた制約を明確にしなければならない。

仕様を記号化ものを獲得するには方法は 2 つが考えられる。

1. 直接形式記述で作成する。

2. 自然言語で書かれたドキュメントを形式化する。

1 つ目は直接形式記述で形式化した仕様書を作成するのは、そこまで難しくない。基礎となる仕様書が存在し無い為、自分で作ったモデルを形式仕様言語で記述するからである。従って、仕様書のなどのドキュメントの内容と合わせる必要が無いからである。

しかし、1 つ目とは逆に 2 つ目の自然言語で書かれたドキュメントを形式化するのは困難である。それは基礎となる標準仕様書がある場合、標準仕様書と対応させながら形式化する必要があるからである。

OSEK/VDX は仕様書が存在する為 2 つ目の形式化の方法を考えなければならない。

1.2.2 形式化の問題点

仕様書と対応した形式化した仕様書を作成する方法が困難な理由は以下の理由があげられる。

- 情報が互いに参照しあっている

仕様書の記述はお互いに参照しながら記述されている。例えば、タスクの状態について記述されている章があったとする。しかし、この章だけタスクの全てが記述できるわけではない。タスクの優先度について書かれている章・タスクと資源について書かれている章などの複数の章を参照してタスクというものが記述できる。従って仕様書のタスクと対応した形式化を行う際、タスクの状態・タスクの優先度・タスクと資源の関係の章の情報を複合して形式化しなければならない。

しかし、複合して形式化を行うと何処の記述と対応して形式化が行われているか分かりにくくなってしまう。

1.2.3　問題点の解決方法

仕様書を分割し形式化しながら形式化した仕様書を獲得する。

この方法より、タスクの状態、タスクの優先度、タスクと資源の関係の章の情報からタスクの形式化を一度にするのではなく、仕様書と同じ様に、タスクの状態、タスクの優先度、タスクと資源の関係の情報をそれぞれ形式化を行う。これにより、お互いの情報を参照せずに形式化を行う事が可能になる。また仕様書と形式化した仕様書の対応した部分の確認が容易になる。

第2章　OSEK/VDX標準仕様書とVDM

2.1　形式仕様記述言語VDM++

VDMは形式手法の一つであり1960年代にIBMウィーン研究所で開発された形式仕様記述言語である。VDM++はそのVDMを基礎にオブジェクト指向拡張を行った言語である。VDM++は主に資源とタスクの関係や制約の観点を関数を用いて形式化していく。これは関数型言語は手続き型の言語に比べてプログラムの理解がしやすいことを利用している。また機能の関数は関数の前に成立しなければ関数を実行できない事前条件、関数が成立した後に成立しなければ関数が実行できない事後条件を記述することが出来る。以下のサイトでVDM++をダウンロードすることができる。 [1], [2] [3]

http://www.vdmtools.jp/modules/tinyd1/index.php?id=1

本研究では、仕様書を形式化する言語として形式仕様記述言語VDM++を使用する。選定理由としては資源とタスクの関係や制約の観点を記述する為の形式仕様書を作成する際の２つの点より決定した。１つ目は、仕様書は実装で使用する値を明確に決めていない為、形式化する時の表現ができない可能性がある。VDM++は引用型という型を明確にせずに記述できる点があげられる。

２つ目は仕様書が「振る舞い」を表現しているか確認できる事である。実際に「検証仕様書」が実行できないと「振る舞い」を確認することは困難である。VDM++はVDMtoolによりインタプリタという実行環境があるので「振る舞い」を確認することができる。

以上の問題を解決できる能力があると考え、本研究ではVDM++を使用する。

VDM++の型

仕様記述言語VDM++は型と関数で仕様書の形式化を行っている。型はVDMではint bool chearなどの型の他に引用型が用意されている。引用型はC言語のシンボルのようなもので型の他に値として扱うことができる仕様書では実装の自由度を保証する為、型の明記を曖昧にしている場合がある。VDM++は引用型を用いることで型を明記していない仕様書の形式化が可能である。

$$types \text{ 参照できる範囲を指定} \tag{2.1}$$
$$\text{データ型} = \text{属する型} \tag{2.2}$$

上式は型定義の構文を示す。参照できる範囲の指定とは型の参照が認められる範囲が記述されている。型の参照は、public private protected ある。public はどのクラスでもこの型を参照してもよい。private は記述されているクラスとその子クラスまでの参照してもよい。protected は記述されているクラスのみ参照してもよい。子の構文はデータ型は属す型で構成されていることを示している。属する型は、基本型と合成型に分けられる。基本型は bool や nat などの型が用意されている。合成型は集合型や列型などがある。

$$typespublic \tag{2.3}$$
$$State =< Ready > \| < Suspended >; \tag{2.4}$$

式 2.4 の State のデータ型を例として挙げる。データ型 State 全てのクラスから参照できる型である。またデータ型 State の属する型は引用型の $< Ready >$ と $< Suspended >$ であることが述べられている。

VDM++の変数と関数

VDM++は変数がありローカル変数とグローバル変数に分けられる。ローカル変数は一回式の実行が終わると値が保存されない。グローバル変数は式が実行された後も値を保存したままである。またローカル変数と違い事前・事後条件などの値の参照も許されている。仕様書の「機能的な記述」は VDM++では関数で形式化する。VDM++は operations と functions がある。functions はグローバル変数にアクセスすることは許されていない。operations はグローバル変数といくつかのローカル変数にアクセスすることが許されている。式を実行した後その値を使用せずに「振る舞い」を表現することは困難である。今回は主に operations を使用して「機能的な記述」の形式化を行った。

operations は以下の構文で示される。

operations 参照出来る範囲の指定
式 : 引数の型 ==> 実行後の型
式 == 関数本体
pre 事前条件
post 事後条件

参照できる範囲の指定とは式の参照が認められている範囲である。式の参照は public private protected ある。public はどのクラスでもこの式を参照してもよい。private は記述

されているクラスとその子クラスまでの参照してもよい。protected は記述されているクラスのみ参照してもよい。引数の型 ==> 実行後の型 は式の型定義を表している。*pre* 事前条件 とは関数が実行される前に成立していなければならない条件である。*post* 事後条件 とは関数が実行された後に成立しなければならない条件である。例として「機能」ActivateTask の「機能的な記述」の一部をあげる。

$$operationspublic$$

$$ActivateTask\_Description(mk\_Task(< TaskIdentifies >, T\_sbj, T\_State))$$
$$== returnmk\_Task(TaskIdentifies\,, T\_sbj, < Ready >)$$
$$preT\_State = Suspended$$
$$postT\_State = Ready$$

式 ActivateTask_Description の VDM++の関数の例である。*public* より全てのクラスから参照できる instance variables と返り値を扱うことのできる式である。$ActivateTask\_Description : TaskID ==> TaskID$ よりこの式はデータ型 TaskID から TaskID の型を返す式である事が分かる。関数本体は、instance variables より引数の値の T_State の部分を¡Ready¿に変換して、その値のローカル変数を返す式である。この式が実行されるのは $preT\_State = Suspended >$ より、引数の T_State が $< Suspended >$ であり、$postT\_State =< Ready >$ よりローカル変数が T_State が $< Ready >$ の時である。

本研究では、グローバル変数を instance vareables、これらの変数を返り値と呼ぶ。

VDM++の実行

VDM++をインタプリタで実行する際まずクラスを生成する必要がある。インタプリタのクラス生成式は

$$cr \text{ インタプリタで認識されるクラスの名前} := new VDM++\text{の記述したクラス}() \quad (2.5)$$

となる。cr インタプリタに VDM++のクラスを定義するコマンドである。

次に関数を実行する場合

$$print \text{ 操作したいクラスの名前.[関数を記述したクラス]‘実行したい関数 (引数)} \quad (2.6)$$

となる。print はインタプリタに結果を表示するコマンドである。

図 2.1 は、ActivateTask を実行している例である。まず

$$cra := newActivateTask() \quad (2.7)$$

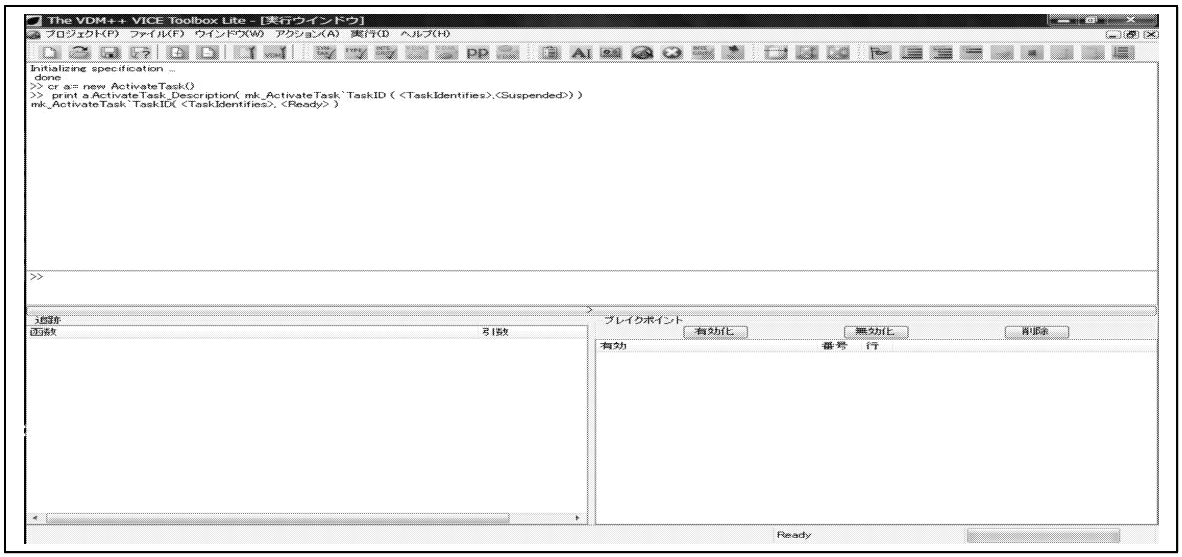

図 2.1: ActivateTask_Description の実行

を実行した。これよりインタプリタに a は ActivateTask のクラスである事が定義された。次に

$$printa.ActivateTask\_Description$$
$$(mk\_ActivateTask\,`TaskID(< TaskIdentifies\,, < Suspended >))$$

を実行した。これは、インタプリタでは、クラス a のローカル変数を変更する Activate-Task_Description が ActivateTask に記述されている TaskID の構造体の引数 $mk\_TaskID(< TaskIdentifies >, < Suspended >)$ で実行された事を示している。実行後

$$mk\_ActivateTask`TaskID(< TaskIdentifies >, < Ready >) \tag{2.8}$$

がインタプリタ表示される。

本研究では 5.1.4 章でインタプリタを使用した「独立ＶＤM記述」と「統合ＶＤM記述」の整合性の確認方法を記述している。

2.2 OSEK/VDX

「OSEK/VDX」とはドイツ・フランスが自動車制御を行うエンジンコントロールユニットで用いる業界標準作成を目標としてプロジェクトで提示されたオペレーションシステムの仕様である。「OSEK/VDX」の主な機能の実現としては処理 (タスク) 管理機能、応用状態 (アプリケーション)、割り込み処理機能、事象 (イベント) 制御機能、資源 (リソース)、警告 (アラーム)、伝言 (メッセージ)、鉤 (フック) ルーチンがあげられる。これらの機能の実現はサービスコールによるタスクの状態の操作で実装されている。

各タスクの状態の内容を以下に示す。

- Suspended 状態

  Run 状態のタスク、タスクが OS に認識された時にこの状態に遷移する。

- Ready 状態

  Susupended 状態、Run 状態のタスクからこの状態に遷移する。Ready 状態のタスクはスケジューラによりそのタスクの優先度毎のレディーキューに格納される。

- Run 状態

  Ready 状態からこの状態に遷移する。

  この状態のタスクは CPU で処理されている。OSEK/VDX はシングルコアの CPU が対象となっている為この状態のタスクは 1 つだけである。

- Waiting 状態

  Run 状態からこの状態に遷移する。

  Run 状態からこの状態に遷移する。この遷移は ExtendedTask のみ遷移が許されている。イベントのタイミングまで CPU を解放してこの状態で待機する。このイベントが実行されると Ready 状態に遷移する。

主なサービスコールの内容は以下に示す。

- DeclaerTask(TaskIdentifer)

  タスクを OS に認識させるサービスコール。

- DeclaerResourece(ResourceIdentifier)

  資源を OS に認識させるサービスコール。

- ActivateTask(TaskID)

  TaskID に該当する Task を Suspended 状態から Ready 状態に遷移させるサービスコール。このサービスコールによりタスクが実行準備状態になる。Ready 状態のタスクはタスクの優先度毎のレディーキューに送られるレディーキューは優先度毎にタスクがレディ状態に遷移した順番で並んでいる。

- Scheduer()

  タスクの優先度やレディーキューの情報より CPU に割り当てるタスクを選択するサービスコール。CPU で処理されている Run 状態のタスクの優先度と最も高いタスクが格納されているレディキューの優先度が比較される。Run 状態の優先度が高いか同じならそのまま Run 状態のタスクが CPU で処理される。Ready 状態のタス

クの優先度が高ければ Run 状態のタスクは Ready 状態に遷移する。そして Run 状態のタスクの優先度のレディーキューに遷移する。選択されたレディーキューの最も古いタスクが Run 状態に遷移して CPU で処理される。。

- TermineteTask()

  Run 状態のタスクを Suspended 状態に遷移させるサービスコール。このサービスコールによりタスクが CPU から解放される。Run 状態のタスクを解放する際には必ずこのサービスコールか ChainTask(TaskID) を選択しなければならない。

- ChainTask(TaskID)

  TaskID に該当するタスクを Ready 状態に遷移させるサービスコール。サービスコール ActivateTask と異なり、該当のタスクが Run 状態の時でも Ready 状態に遷移する。Run 状態のタスクを解放する際には必ずこのサービスコールか TerminateTask() を選択しなければならない。

- GetResourece(ResID)

  ResID に該当する資源を Run 状態のタスクが獲得するサービスコール。資源にも優先度があり、資源を獲得したタスクは資源と同じ優先度となる。また資源の優先度より優先度が低いタスクの資源の獲得は禁止されている。仕様書に資源は厳密に LIFO のように獲得・解放しなければならないと決められている。

- ReleaseResourece(ResID)

  ResID に該当する資源をタスクから解放する資源はタスク毎に LIFO [1]の様に管理されてる。仕様書に資源は厳密に LIFO のように獲得・解放しなければならないと決められている。

「OSEK/VDX」はアプリケーションの設定記述を行う為の専用言語である OIL(OSEK Implementation Language) がある。OIL で記述したアプリケーション設定ファイルをシステムジェネレータに通すと例えば C 言語のソースファイルを出力する。OSEK は C 言語記述以外認めていないわけではないので他の言語のソースファイルを出力するシステムジェネレータがあってもよい。この OIL により様々な実装が作られる。従って作成された様々な OSEK/VDX の実装が OSEK 仕様に準拠しているか確認する為の方法が求められている。

2.3 OSEK/VDX 仕様書

OSEK/VDX の仕様書は以下のサイトで入手可能である。[4]

[1] stack

固有の名称の説明 ⟶

機能の振る舞いの側面 ⟶

4.2.2 Basic tasks

The state model of basic tasks is nearly identical to the extended tasks state model. The only exception is that basic tasks do not have a *waiting* state.

running In the *running* state, the CPU is assigned to the task, so that its instructions can be executed. Only one task can be in this state at any point in time, while all the other states can be adopted simultaneously by several tasks.

ready All functional prerequisites for a transition into the *running* state exist, and the task only waits for allocation of the processor. The scheduler decides which *ready* task is executed next.

suspended In the *suspended* state the task is passive and can be activated.

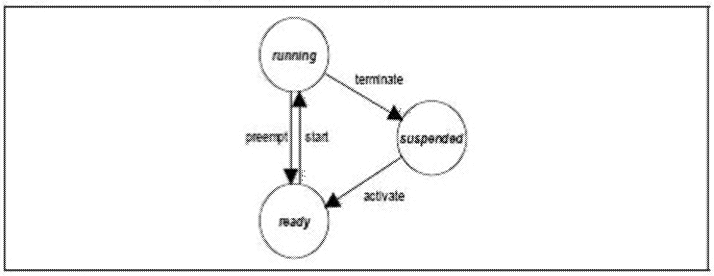

Figure 4-3 Basic task state model

Transition	Former state	New state	Description
activate	*suspended*	*ready*[4]	A new task is set into the *ready* state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction.
start	*ready*	*running*	A *ready* task selected by the scheduler is executed.
preempt	*running*	*ready*	The scheduler decides to start another task. The *running* task is put into the *ready* state.
terminate	*running*	*suspended*	The *running* task causes its transition into the *suspended* state by a system service.

Figure 4-4 States and status transitions for basic tasks

図 2.2: 仕様書の BasicTask の章

http://portal.osekvdx.org/index.php?option=com_frontpage&Itemid=1

OSEK/VDX は全 P86 ページで 142 章で構成されている。
OSEK/VDX の仕様書の記述内容を以下に示す。

1. OSEK/VDX は機能の振る舞いをある側面について説明や、固有の名称や説明が記述されている

2. 実装の機能を実現するシステムコールの構文・引数・返り値・システムコールについての説明・エラーについて記述されている

図 2.2 は 1 つ目に該当する部分である。図では BasicTask の runing や ready などの固有の名称の説明は自然言語で説明されている。

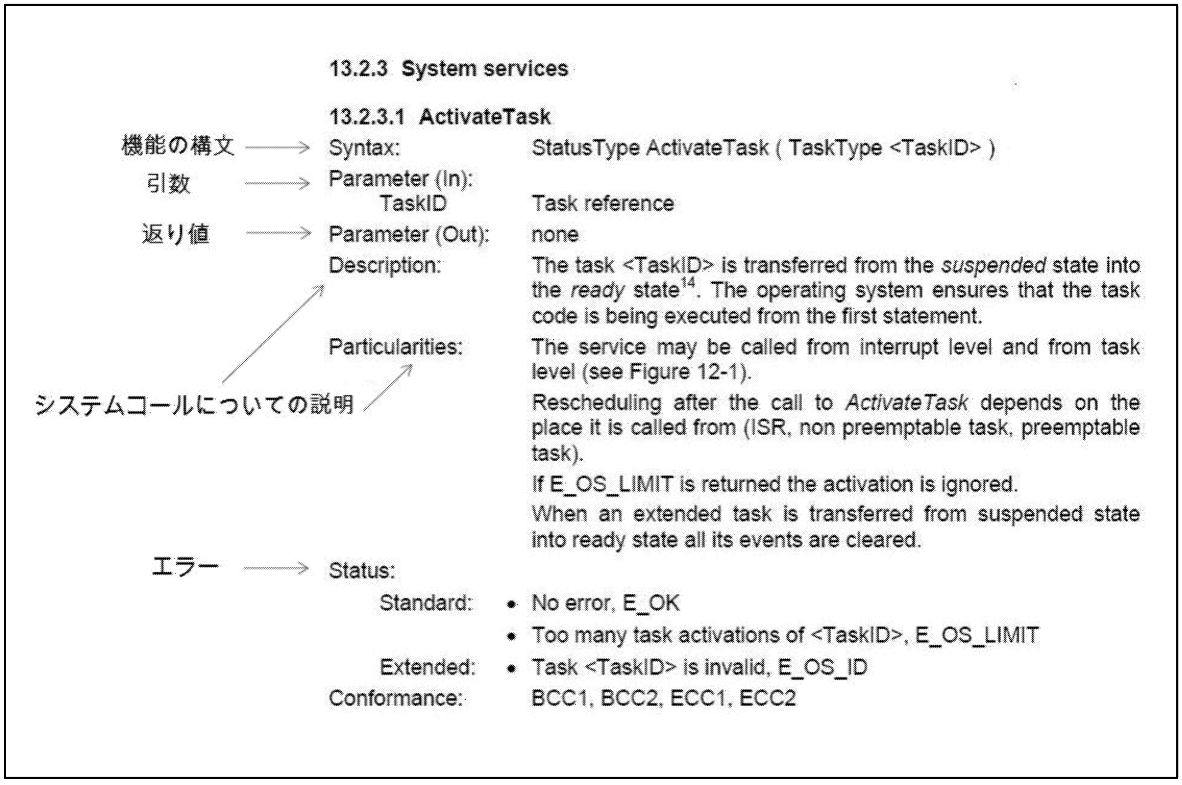

図 2.3: 仕様書の 2 つ目に当たる ActivateTask の章

このBasicTaskの章は固有の説明や遷移するイベントをタスク状態の観点から自然言語や図を用いて記述している。

図 2.3は仕様書の２つ目に該当する部分である。

図 2.3のActivateTaskの章はサービスコールの観点から必要な機能の構文や引数や返り値やエラーの情報を自然言語で記述している。しかし、この部分だけではActivateTaskの全てについて理解することは困難である。TaskIdentiferやISRなどの固有の名称は他の章の記述を参照しなければ理解できないからである。

2.3.1 OSEK/VDXの仕様書に関する問題点の例

1.2.2章の問題点をOSEK/VDXの仕様書を使用して例をあげていく。

情報が互いに参照

BasicTaskの章とTaskpriorityの章を検証仕様書に変換していく例をあげる。タスクのデータ型に着目して説明していく。仕様書のBasicTaskの章を検証仕様書に変換する。BasicTaskの章はタスクの状態の観点から記述している。従ってタスクのデータ型はタスクの状態だけの情報を持つ。そしてこのデータ型を基に関数によって仕様書の形式化を記述を行う次に検証仕様書にTaskpriorityの章の情報を追加する。Taskpriorityの章はタスクの優先度の観点から記述している。従ってTaskpriorityの章にはタスクのデータ型に優先度を持たなければならないことが分かる。Taskpriorityの章の情報によりタスクのデータ型が変わったのでBasicTaskの章を形式化した部分を見直さなければならない。

また仕様書の内容を把握して検証仕様書を作成する場合でもOSEK/VDXの仕様書は142章もあるので全てを把握しながら作成するのは困難である。

この様にOSEK/VDXの仕様書においても、分散している様々な観点からの情報から一度に仕様書の形式化と様々な観点を組み合わせて検証仕様書を作成していくのは困難である事が分かった。

第3章　形式仕様書の獲得方法

3.1　情報が互いに参照に対する解決方法

2.3.1 章に対する解決方法は以下の様に提案する。

- 仕様書を章ごとに分割して個々に形式化

仕様書は１つのデータ型を様々な側面から記述している。仕様書を章ごとに分割して形式化を行う事で、側面毎に形式化をすることが出来る。

これにより得られる形式化した仕様書は、仕様書と１対１で形式化されているはずである。

仕様書と１対１で形式化されていることを行う為には以下の確認が必要である。

- 仕様書と形式化された仕様書の関係
- 仕様書の一貫性

3.2　仕様書と形式化された仕様書の関係

仕様書と形式化された仕様書は１対１で対応しているはずである。本研究では仕様書のどの部分を形式化を行ったかを示す為対応関係表を作成する。対応関係表は形式化を行った元の仕様書の文章と形式化された文章が記述されており、対応がとれる記述になっていればよい。対応関係表は以下の表の様に記述する。

対応関係表を確認することにより、仕様書のどの部分が形式化されたか分かる。

3.3　仕様書の一貫性

仕様書が１対１で形式化されたのであるならば形式化された仕様書はデータ構造を様々な側面から記述したものになっている。従って、形式化された仕様書からデータ構造を作成する事が可能である。このデータ構造を使って振る舞いを持った仕様書を作成する事が出来る。

もし振る舞いを持った仕様書の作成が出来ないのであるならば以下の可能性がある。

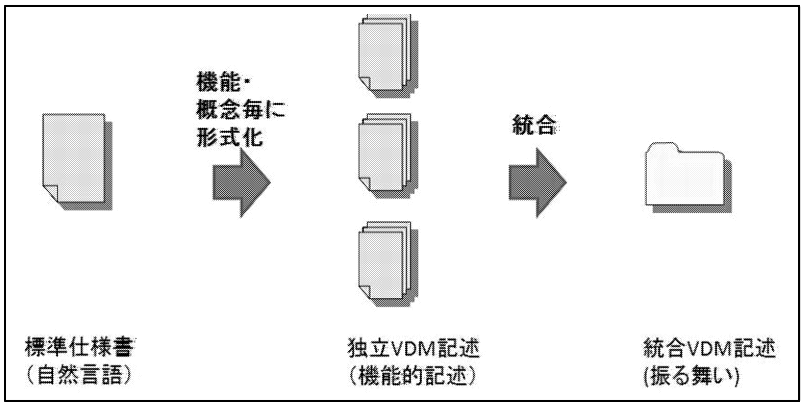

図 3.1: 提案手法の全体図

- 1 対 1 で形式化されていない
- 形式化の内容が間違っている

1 対 1 の形式化が出来ていない場合、仕様書の情報が欠落している可能性がある。その為、振る舞いに変換する為の情報が無い為、振る舞いを持った仕様書に変換できない。

また、形式化した内容が間違っている場合も誤った振る舞いを持った仕様書が出来てしまう。

3.4 提案手法の全体図

解決方法から提案手法の全体図を図 3.1 を示す。
仕様書の形式化と確認方法は以下の様に行われる

1. 独立ＶＤＭ記述の作成
2. 対応関係表の作成
3. 統合ＶＤＭ記述の作成
4. 一貫性の確認

3.4.1 独立ＶＤＭ記述の作成

独立ＶＤＭ記述の作成の目的は以下である。

- 仕様書のドキュメントの形式化

仕様書のドキュメントの形式化は形式化された記述と 1 対 1 にならなければならない。本研究では、仕様書を章ごとに形式化を行う事で対応する。

形式化の手順は以下のように行う。

1. 章ごとに区切る
2. 型定義
3. 操作の定義

章ごとに区切る

仕様書の記述同士の参照を防ぐ為に仕様書の記述を分割する。分割する条件として以下の点に注意した。

- 記述の途中で区切らない

仕様書の記述を途中で区切ってしまうと形式化で出来ない。

本研究では、仕様書の記述を途中で区切らない分け方として章ごとに区切って形式化を行った。

型定義

仕様書の記述を形式化する為には型定義をしなければならない。型定義は、仕様書の記述に用いられる数値の範囲を決定する。形式化された仕様書の中には同じ型定義を用いて記述することは出来ない。仕様書の章の記述は、1 つの機能を同じ観点から記述している。従って、章ごとの形式化を行う際は、同じ型定義を使用する可能性が高い。これを解決する為、本研究では章ごとに区切った中の型定義においては共有して使用するものとする。

操作の定義

操作の定義とは仕様書の記述を形式化の本体に当たる部分である。VDM では仕様書の形式化を行う為、自然言語で書かれた操作を関数に変換して形式化を行う。

3.4.2 対応関係表

仕様書のどの部分を形式化したのか確認しづらい為、対応関係表を作成する。対応関係表は以下の記述がなされている。

- 形式化する仕様書の記述

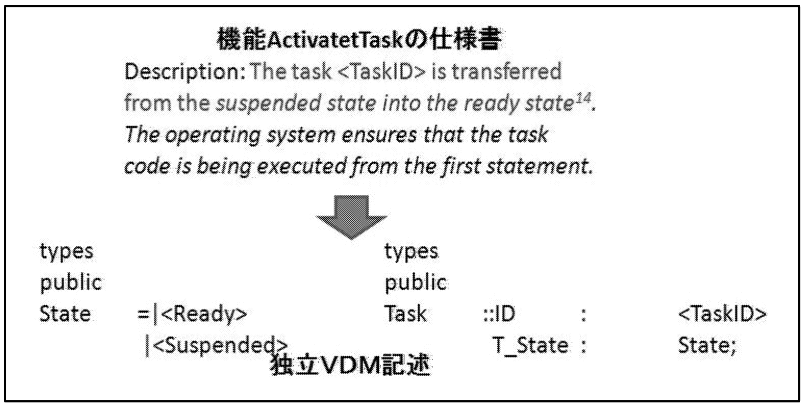

図 3.2: 「独立ＶＤＭ記述」の型定義の変換

図 3.3: 独立ＶＤＭ記述：機能の形式化

- 形式化した仕様書の記述
- 形式化した章

本研究では以下の様に作成した。

3.4.3 統合ＶＤＭ記述の作成

統合ＶＤM記述の作成の目的は、一貫性を確認する為に作成する。
独立ＶＤM記述で作成した形式化に問題がなければ、そこから振る舞いを持つ形式化された仕様書が作成できる。
統合ＶＤM記述の作成は以下の手順で作成される。

1. 型定義の統合
2. 型定義から変数
3. 独立ＶＤM記述の情報から振る舞い

3.4.4 型定義の統合

型定義の統合は独立ＶＤM記述で別々に記述されたデータ型の情報が同じものを示している場合統合する。
実装からみれば同じデータ型であるが、独立ＶＤM記述でそれぞれ記述されたデータ型の情報の内容は異なっている。これは、章ごとに注目している Task の情報が異なっている為である。
一つの例として Task を例に挙げる。ActivateTask の章の Task のデータ型は State と TaskID の情報がある。Taskpriority の章の Task のデータ型は State と priority の情報がある。
これらは Task のデータ型であるがそれぞれ異なった情報を持っている。この２つの章を統合して得られる Task の情報は State と TaskID と priority である。
本研究ではこの様なデータ型を統合して統合ＶＤM記述の作成に必要な型を作成する。

3.4.5 型からの変数

統合した型から変数を作成する。これは振る舞いを作る際、操作した後のデータ型を保存しておく必要があるからである。
本研究では型から変数を作る際、型の集合を用いて変数を作成した。

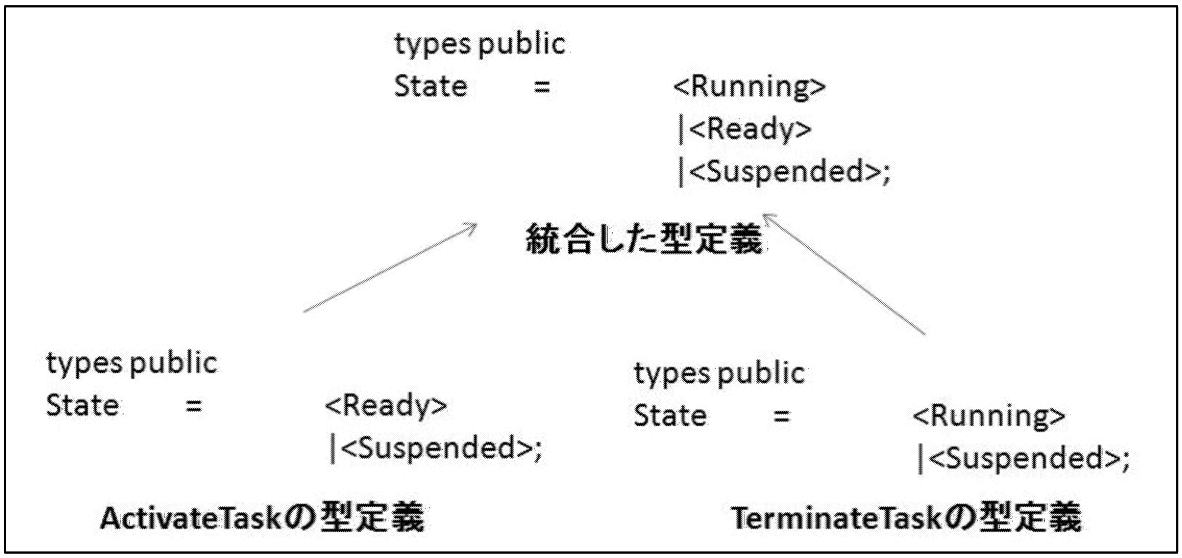

図 3.4: 「統合ＶＤＭ記述」における型定義の統合

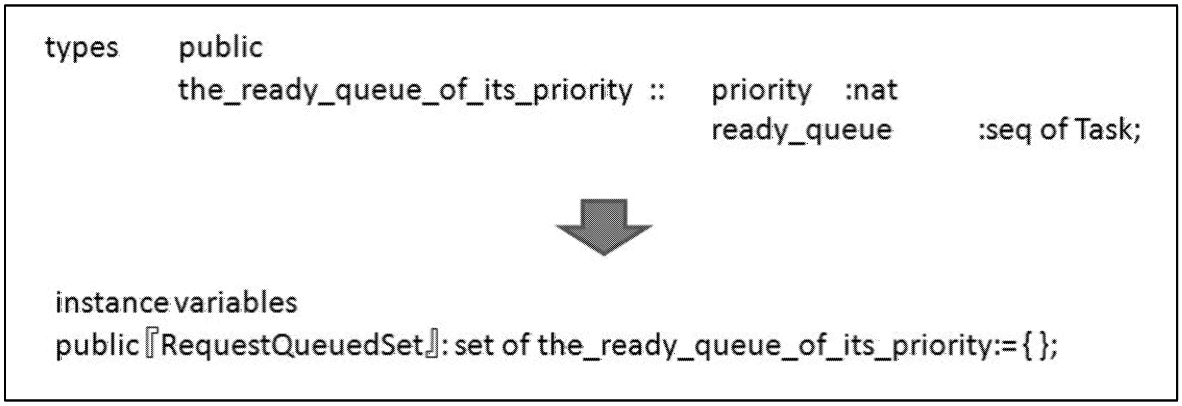

図 3.5: 「統合ＶＤＭ記述」における型定義から変数の例

3.4.6　独立ＶＤＭ記述の情報から統合ＶＤＭ記述の作成

独立ＶＤＭ記述の情報から統合ＶＤＭ記述を作成する。

統合ＶＤＭ記述は形式化された振る舞いで記述されている。

しかし、統合ＶＤＭ記述を作る為には独立ＶＤＭ記述の情報を組み合わせて作成する。独立ＶＤＭ記述の情報はそれぞれ別々に記述されている為独立ＶＤＭ記述同士の関連が分かりにくい。

その為振る舞いを作成しづらい。

本研究では独立ＶＤＭ記述に記述された事前・事後条件を用いて、独立ＶＤＭ記述の関連を付ける。

章はそれぞれある動作に注目して記述されている。独立ＶＤＭ記述は形式化されており、操作の定義に事前・事後条件が含まれることがある。

事前条件は、操作の定義が行われる前に成立していなければならない条件である。事後条件は、操作の定義が行われた後に成立していなければならない条件である。

これより、同じ事前・事後条件が含まれている章は、ある操作の定義を別の観点から記述されている可能性が高い。

OSEK/VDX はタスクの状態の制約が多い為、本研究ではタスクに関する事前・事後条件を基に操作の定義をグループ分けを行う。

3.4.7　一貫性の確認

独立ＶＤＭ記述と統合ＶＤＭ記述を用いて一貫性の確認を行う。

一貫性とは、仕様書と一対一で形式化出来た場合、形式化した仕様書から全体の振る舞いが作成できる事である。

本研究では独立ＶＤＭ記述が一貫性を持っている事を確認する為に統合ＶＤＭ記述を作成した。

しかし、統合ＶＤＭ記述が独立ＶＤＭ記述の操作の定義を含んでいる事を確認しなければならない

VDM++は操作の定義をインタプリで実行することが可能である。

独立ＶＤＭ記述と統合ＶＤＭ記述のそれぞれ引数の対応関係があるとする。それらの引数を用いて独立ＶＤＭ記述と統合ＶＤＭ記述をインタプリタで実行する。それらの返り値の対応関係があるのであれば統合ＶＤＭ記述は独立ＶＤＭ記述の操作の定義を含んでいる事が確認できる。

第4章　OSEK/VDXの形式化

本研究では、提案手法を OSEK/VDX に適用した。本研究で今回形式化を行ったのは以下である。

- ActivateTask
- ChainTask
- DeclearTask
- GetResource
- Scheduler
- Taskpriority
- OsekPriorityCeillingProtocol
- RereaceResource
- TerminateTask
- BasicTask
- ExtendedTask
- Legitimacy of calls
- Activate a Task
- Task Swiching Mechanism
- Full preemptive Scheduling
- Mixed preemptive scheduling
- Non preemptive Scheduling
- Termination of Tasks

- InternalResourece
- Structur of the description [1]

4.1 ActivateTaskの形式化

ActivateTaskの章の独立ＶＤM記述の作成を行った。
要求仕様書には以下の記述がある

```
Syntax: StatusType ActivateTask ( TaskType <TaskID> )
Parameter (In):
TaskID Task reference
Parameter (Out): none
```

これはActivateTaskの構文・引数・返り値について記述されている。
構文は$ActivateTask(TaskTypeTaskID)$である。引数はタスクを参照するTaskIDが用いられており、返り値は何も返さない。
この形式化については構文と引数と返り値に着目して行った。引数はTaskIDである。このTaskIDは仕様書には具体的なデータ型は記述されていない。この場合、VDM++では引用型を用いて表す。
次に返り値を考える。返り値はnoneなので、返り値が無いと事を表現した。
これより、この部分に対応する独立ＶＤM記述は以下のようになる。

```
operations public
Activate_Task: TaskID ==>()
Activate_Task(TaskID) == return ;
```

次にActivateTaskのDescriptionについての記述である。

```
Description: The task <TaskID> is transferred from the suspended state
into the ready state14.
```

この記述はTaskIDのタスクがSuspended状態からReady状態に遷移することが記述されている。
これはタスクがTaskIDの情報とSuspended状態とReady状態の情報を持つ事が分かる。従って型定義は以下のものを作成した。

[1]起動順の確認の為1部しか形式化していない

```
types public
Task ::ID : <TaskID>
  T_State : State;

types public
State = <Ready>
|<Suspended>;
```

Task は ID と State の型定義を持つ。ID は TaskID の事であり、これは TaskID の引用型を用いて表した。State は Ready 状態と Susupeded 状態を持つことがわかる。Ready 状態は *Ready Suspended* 状態は *Suspended* に対応してそれぞれ型定義を決定した。

つぎにタスクが Suspended 状態から Ready 状態になる操作の定義は以下のように記述した。

```
operations public
ActivateTask_Description: Task ==>Task
ActivateTask_Description( mk_Task ( <TaskID>,T_State) ) ==
return mk_Task (<TaskID>,<Ready>)
pre T_State = <Suspended>
post T_State = <Ready>
```

タスクの Description は Task の構造体の引数から Task の構造体の T_State を $Ready$ 状態を返す関数で記述した。また、この関数の事前条件として引数の T_State が $Suspended$ とし、事後条件の返り値の T_State が $Ready$ であるという制約を記述した。次に

```
If E_OS_LIMIT is returned the activation is ignored.
```

これは ActivateTask が発行されてもタスクが起動しなかった場合 E_OS_LIMIT を返すことが記述されている。

```
types public
TaskState = <ignore_activation>

operations public
ActivateTask_Particularites2: TaskState ==>Status
ActivateTask_Particularites2(TaskState) == return <E_OS_LIMIT>
```

ignore actiovation の具体的な状態が情報不足の為記述できない。その為、ignore activation を引用型として扱った。

これは Task が起動できなかった場合、E_OS_LIMIT を返すという記述を関数で操作の定義をしている。次に以下の記述がある。

```
When an extended task is transferred from suspended state
into ready state all its events are cleared
```

これはExtendedTaskの場合Suspnededから Ready状態に遷移する場合イベントを全て消去するという記述である。

この形式化については以下のように記述した。

```
types public
ExtendedTask = State*Event_Set

types public
Event = <event>

types public
Event_Set = set of Event

operations public
ActivateTask_Particularites3:ExtendedTask ==>ExtendedTask
ActivateTask_Particularites3( mk_(State, Event_Set)) ==
 return mk_( State, {})

pre Stete = <Suspended>
post State = <Ready>
```

ExtendedTaskはStateとEventを持つことが仕様書より確認できる。従ってExtended-Taskのデータ型にStateの型とEventSetを定義した。EventSetはEventの型の集合である。Eventは1つなのか複数なのか分からなかったので集合で表した。これにより複数の場合でも表現できる。

Eventの具体的な型は明記されていない為引用型で表現した。

操作の定義はExtendedTaskの $Event\_Set$ を空集合にして値を返す事によって表現した。また制約として、事前条件はStateが $Suspneded$ とし、事後条件はStateは $Ready$ をつけることにより形式化した。次に以下の仕様書の記述がある。

```
Status:
Standard: ? No error, E_OK
? Too many task activations of <TaskID>, E_OS_LIMIT
Extended: ? Task <TaskID> is invalid, E_OS_ID
```

これはエラーについての記述がなされている。エラーが無ければ E_OK、起動するタスクの数が多いのであれば E_OS_LIMIT、タスクのIDが正しくなければ E_OS_ID を返すという記述がある。

この記述は以下のように形式化を行った。

```
types public
Status = <E_OK>
|<E_OS_LIMIT>
|<E_OS_ID>

types public
Status_Case = <NoError>
|<ManyActivationsOfTaskID>
|<TaskIDIsInvalid>

operations public
ActivateTask_Status: Status_Case ==>Status
ActivateTask_Status(Status_Case) == cases Status_Case:
<NoError>->return <E_OK>,
<ManyActivationsOfTaskID>->return <E_OS_LIMIT>,
<TaskIDIsInvalid>->return <E_OS_ID>
end
```

まず Status のデータ型はエラーを示している。これは具体的な値が示されていないので、引用型を用いる。

次に各エラーが起こる状態を Status_Case で型定義した。エラーが起こる状態の具体的な状態を値を用いて表す事は困難である。従って状態を引用型を用いて定義した。

これより、エラーが起きる状態とエラーの値が決まったので、エラーが起きる状態に対応するエラーを返す関数を操作の定義として決定した。

4.2 DeclearTask の形式化

DeclearTask の章の形式化を行った。DeclearTask のサービスコールについての記述されている。DeclareTask は、Constructional elements に該当する部分である。これは引数のみをとり、返り値は明記されていない。これは Constructional elements が呼び出されると、システムに情報が加わる。この加わった情報は常に正しいと OS 側で判断される為、エラーや返り値は用意されていない。

仕様書には以下の記述がある。

```
Syntax: DeclareTask ( <TaskIdentifier> )
Parameter (In):
TaskIdentifier Task identifier (C-identifier)
```

これは DeclareTask の構文と引数について記述されている。

この記述は以下のように形式化を行った。

```
types public
TaskIdentifier = <C_identifier>

operations public
DeclareTask: TaskIdentifier ==>()
DeclareTask(TaskIdentifier) == return undefined ;
```

TaskIdentifier は C 言語の様な定義がなされることが記述されている。しかし C 言語の様な定義を明確に値に示すことは困難である。TaskIdentifier の型定義は引用型をもちいて表現した。

DeclareTask の構文と引数について形式化を行った。

TaskIDentifier を引数に DeclareTask の関数を作成した。仕様書では返り値については明記されていない。そこで VDM では、仕様書の未定義である部分に対して undefined が用意されている。これを用いて DeclareTask の操作の定義を作成した。

```
Description: DeclareTask serves as an external declaration of a task.
The function and use of this service are similar to that
of the external declaration of variables.
```

DeclaerTask は Task を OS で認識させるサービスコールであることが記述されている。

形式化は以下のように行った。

```
types public
TaskIDset : set of TaskIdentifier ;

operations public
DeclareTask_Discription: TaskIdentifier ==>TaskIDset
DeclareTask_Discription(TaskIdentifier) ==
return TaskIDset union {TaskIdentifier}
```

OS が TaskIdentifier を認識する為に TaskIDset という TaskIdentifier の集合のデータ型を定義した。TaskIDset の集合に入っている TaskIdentifier は OS で認識されているという意味を持っている。

次に操作の定義を用いて TaskIDset に TaskIdentifier を加える関数を定義した。これより DeclareTask が TaskIdentifier を OS に認識させるサービスコールであることを表した。

4.3 TerminateTask の形式化

TerminateTask の章の形式化を行った。TerminateTask のシステムコールについて記述されている。

仕様書には以下の記述がある。

```
Syntax: StatusType TerminateTask ( void )
Parameter (In): none
Parameter (Out): none
```

TerminateTaskの構文と引数・返り値について記述されている。TerminateTaskの引数・返り値はともに無い。

以下の様に形式化を行った。

```
operations public
Terminate_Task:() ==>()
Terminate_Task() == return
```

引数と返り値は無い為、ともにvoidで表した。次に以下の様な記述がある。

```
escription: This service causes the termination of the calling task.
The calling task is transferred from the running state
into the suspended state15.
```

TerminateTaskの機能の説明について記述されている。TerminateTaskはタスクをRunning状態からSuspended状態に遷移させる事が記述されている。

この記述に対しては以下の様に形式化を行った

```
types public
Task :: T_State : State;
types public
State = <Running>
|<Suspended>;

operations public
TerminateTask_Description: Task ==>Task
TerminateTask_Description( mk_Task (T_State)) ==
return mk_Task (<Suspended>)

pre  T_State = <Running>
post T_State = <Suspended>
```

StateはTaskの状態を表しており、Running状態とSuspended状態を持つことが分かる。TaskはStateを持つこと。従ってTaskとStateの型定義はこの様に定義した

TerminateTaskの説明は操作の定義より決定した。

TerminateTaskの関数よりタスクからタスクのStateをSuspended状態にして返す関数を用意した。

また、制約として Task の State は事前条件は Running 状態、事後条件は Suspneded 状態であることを定義した。

第5章　実験：統合VDM記述

実際に提案手法で示した手順をOSEK/VDXを用いて実験を行った。今回の形式化はタスクに関する状態が変化するに着目して全体の振る舞いを作成する。

5.1　実験：統合VDM記述作成手順

5.1.1　「独立ＶＤＭ記述」で形式化していない所を形式化

「統合VDM記述」作成の段階は別の「独立VDM記述」を参照しながら調整・統合を行う。まず、「独立VDM記述」の情報不足で形式化できなかったところを形式化する。この形式化は「独立ＶＤＭ記述」と同じ方法で行う。

5.1.2　型の統合

それぞれ「独立VDM記述」より型を統合する。「独立ＶＤＭ記述」で定義した同じデータを表した型の情報を組み合わせる。これは全ての型において行う。同じ型の要素を持つ型は同じデータを表している事が多い。同じ型に注目して統合を行うとスムーズに作成が出来る。同じデータを表している型の要素を統合する際は、同じ要素の型を一つにまとめる。この型の統合により、「振る舞い」に必要なinstance variablesを作る為の型の情報を整理する事が可能になる。図 5.1は実際に「独立ＶＤＭ記述」のActiveTaskとTerminateTaskの型定義を統合した例の図である。ActivateTaskのStateの型定義は$<$ *Ready* $>$ と $<$ *Suspended* $>$ である。TerminateTaskのStateの型定義は$<$ *Suspended* $>$ と $<$ *Ready* $>$ である。このStateはともにタスクの状態におけるデータの型定義を表している。しかし、２つ型定義は$<$ *Suspended* $>$ という同じ型が存在する。このような場合は１つの$<$ *Suspended* $>$ を一つにして統合される。

5.1.3　統合した型から変数の作成

「統合ＶＤＭ記述」で作成する「振る舞い」は「独立ＶＤＭ記述」で記述した「機能的な記述」から作成される。「振る舞い」を表現するにはプログラム言語の変数に当たる部分が必要である。VDM++ではプログラムの変数に相当するinstance variablesを使用す

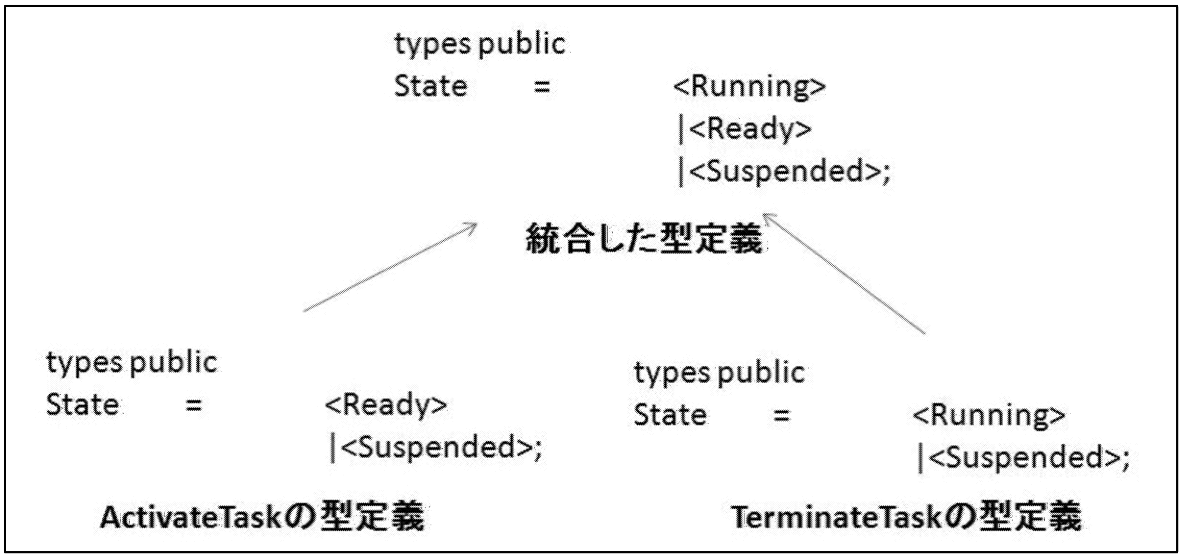

図 5.1: 「統合ＶＤM記述」における型定義の統合

る。instance variable は変数の型を定義しなければならない。今回は「機能的な記述」から「振る舞い」を作る方法として「機能的な記述」の返り値から「振る舞い」を実現する方法で変換した。従って inctance variavle は「独立ＶＤM記述」で作成たデータ型から作成するのが望ましい。「機能的な記述」を保持する instance variables は「独立 VDM 記述」で統合したデータ型の集合で構成したものを使って表現した。

図 5.2 は、レディキューにおける型定義から変数に変換する例である。

型定義 *the_ready_queue_of_its_priority* は統合したレディーキューの型である。統合時の各優先度毎レディーキューの集合を『RequestQueuedSet』と変数を命名した。今回の例『RequestQueuedSet』は型定義 t*the_ready_queue_of_its_priority* の集合が型で、集合は空集合で有ることを示している。

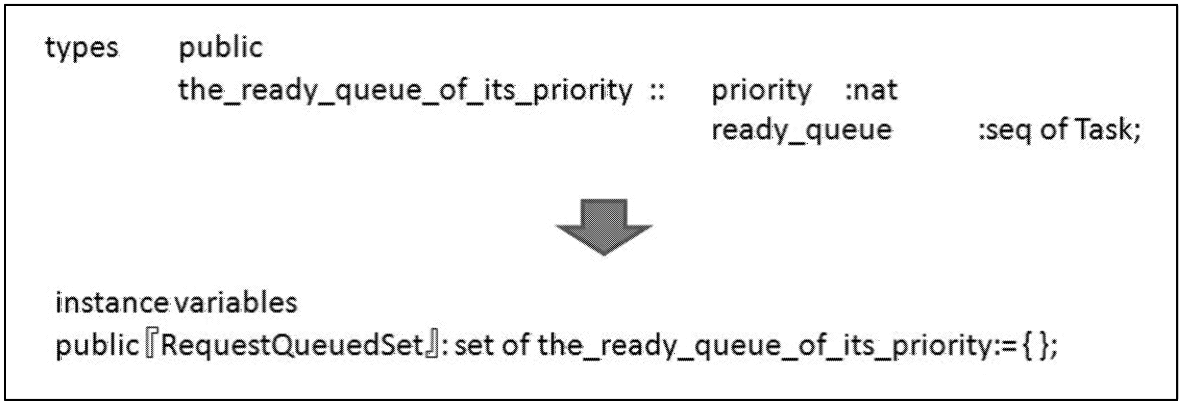

図 5.2: 「統合ＶＤM記述」における型定義から変数の例

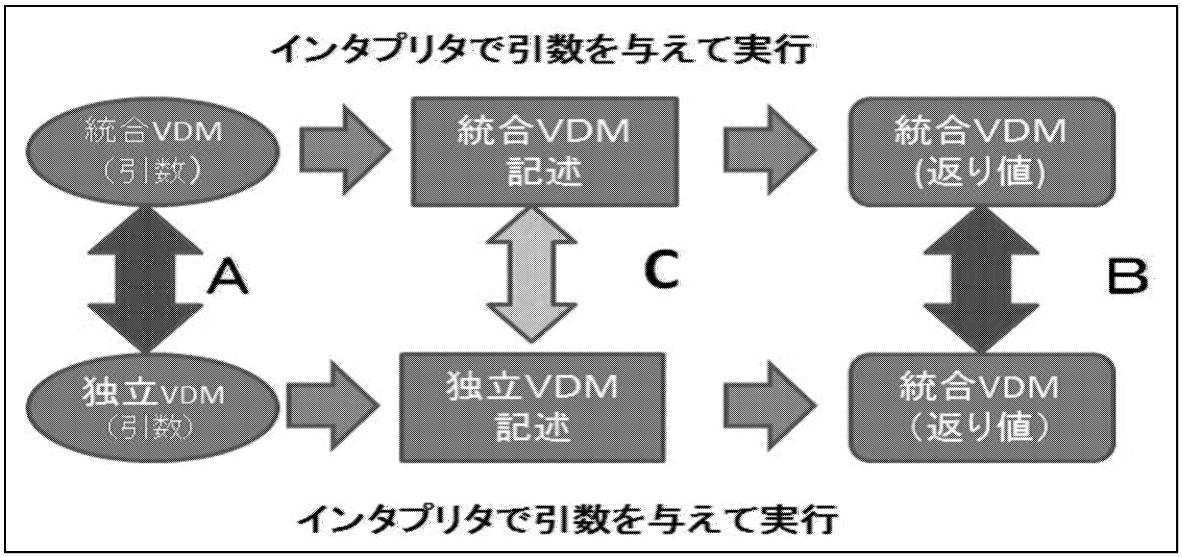

図 5.3: 「独立ＶＤM記述」と「統合ＶＤM記述」の整合性

5.1.4 独立 VDM 記述と統合ＶＤＭ記述の一貫性

統合ＶＤM記述を作成することによりに独立ＶＤM記述の一貫性の確認を行った。しかし、独立ＶＤM記述と統合ＶＤM記述の一貫性を明示的に示していない問題点がある。この問題点を解決する為には全体の振る舞いの統合ＶＤM記述が独立ＶＤM記述を満たしているか確認しなければならない。そこで VDMtool のインタプリタによる実行機能を利用して確認する。図 5.3 に流れを示す。独立ＶＤM記述の操作の定義は引数を実行すると返り値を返す。独立ＶＤM記述の引数が統合ＶＤM記述の操作の引数の対応関係を図中の A とする。独立ＶＤM記述の返り値が、統合ＶＤM記述の instance variables の値と対応関係を図中の B とする。独立ＶＤM記述と統合ＶＤM記述の対応関係を図中の C とする。もし、A が成り立つ引数を統合ＶＤM記述と独立ＶＤM記述の VDM 実行環境のインタプリタで実行する。この時 B が成り立つ返り値のであれば C が成り立つ事が言える一つの方法であるといえる。

独立 VDM 記述：ActivateTask と統合ＶＤＭ記述:Activate_Task の例

実際に「統合ＶＤM記述」ActivateTask が「独立ＶＤM記述」の ActivateTaskDisctription の「機能的記述」の一貫性を例に考える。

引数の対応関係

図 5.4 は、それぞれの記述のタスクの型である。引数の対応関係を考える場合引数の型を見るのが理解しやすい。「統合ＶＤM記述」は ActivateTask(ID) で ID が引数である。

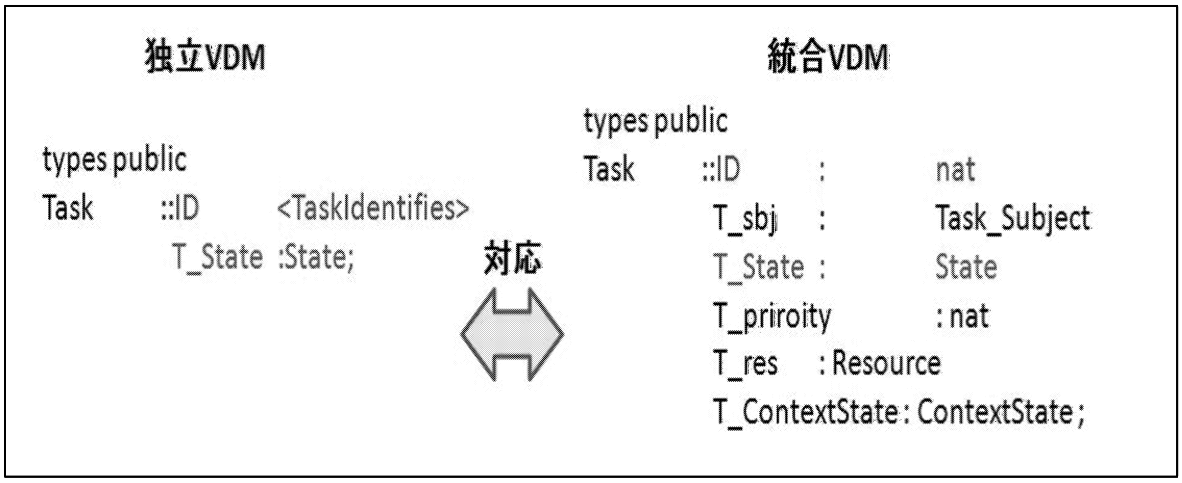

図 5.4: 「独立ＶＤＭ記述」タスクの型と「統合ＶＤＭ記述」のタスクの型

ActivateTask の引数 ID に該当するタスクについて操作を行うので、その ID に該当するタスクが「独立ＶＤＭ記述」のタスクと同じで有れば引数の対応関係がとれる。

ID に該当するタスクについて考える。「独立ＶＤＭ記述」の ActivateTask で ID について具体的に触れられていないため引用型である $< TaskIdentifer >$ を使用した。しかし「統合ＶＤＭ記述」では「振る舞い」を表現できるように引用型から nat 型に定義した。この事より、$ID =< TaskIdentifies >$ と $ID = nat$ は対応関係であることが考えられる。

タスクの状態を示す型 State は同じ意味で型定義されているのでそのままで考える。

以上の事より「独立ＶＤＭ記述」と「統合ＶＤＭ記述」のタスクというのは $ID =< TaskIdentifies >$ と $ID = nat$ が対応関係ある事と と $T\_State = T\_State$ であるならば「統合ＶＤＭ記述」と「独立ＶＤＭ記述」のタスクと対応関係があるとする。

「統合ＶＤＭ記述」の型は他にも T_pritoriy などあるが「独立ＶＤＭ記述」の型では定義されていないので値が変化しても、上で示した同じタスクである条件を満たせば対応関係があるタスクとする。

このことから、「独立ＶＤＭ記述」と「統合ＶＤＭ記述」の具体的な引数を考えていく。

今回の「独立ＶＤＭ記述」の引数は $mk\_Task(< TaskIdentifer >, < Suspended >)$ である。

次に「統合ＶＤＭ記述」のタスク $mk\_impvdm`Task(1, < BasicTask >, < Suspended >, 2, [], < Oteher >)$ を用意する。

ID にあたる部分は 1 であり $< TaskIdentifer >$ と対応関係があるとする。状態を示す値は $< Suspended >$ である。「独立ＶＤＭ記述」で用意した引数と同じタスクの状態である。従ってこの「統合ＶＤＭ記述」のタスクは「独立ＶＤＭ記述」の引数と同じであるということが分かる。

「統合ＶＤＭ記述」の ActiveteTask の引数は「統合ＶＤＭ記述」で用意したタスクを動かすものでなければならない。「統合ＶＤＭ記述」の ActiveteTask(ID) は ID を指定し

てIDに該当するタスクを起動する。従って「統合ＶＤＭ記述」の引数は1である。

返り値の対応関係

次にこれを実行する。実行は2.1の方法でインタプリタを使用して実行した。

これをそれぞれインタプリタで実行して出てくる値は「独立ＶＤＭ記述」は$mk_Task(< TaskIdentifer >, < Ready >)$。

「統合ＶＤＭ記述」は$『Taskset』 = \{mk_impvdm`Task(1, < BasicTask >, < Ready >, 3, [], < Other >)\}$。

「統合ＶＤＭ記述」の『Taskset』はOSEK/VDXが認識しているタスクの集合である。今回タスクの集合の要素でIDが1になっているタスクがActiveteTaskで操作されたタスクに該当する。

最後に返り値の対応関係についてみていく。対応関係であるタスクの条件$< TaskIdentifer >= 1$は満たされている。またタスクの状態も「独立ＶＤＭ記述」と「統合ＶＤＭ記述」ともに$< Ready >$になっているのでこの返り値は対応関係があるといえる。

このことより「独立ＶＤＭ記述」のActiveteTaskDiscriptionと「統合ＶＤＭ記述」のActiveteTaskは対応関係が有る。したがってActiveteTaskDiscriptionとActiveteTaskの整合性はとれている事が言える。

第6章　評価・考察

提案手法より、仕様書から一対一に形式化した独立ＶＤM記述を獲得した。

１対１で仕様書と独立ＶＤM記述が対応しているか明確にする為、対応関係表を作成した。

また、独立ＶＤM記述の一貫性を確認する為に統合ＶＤM記述を作成した。

今回仕様書から独立ＶＤM記述に形式化する際の正しさを示す方法については言及できなかった。今回は対処として仕様書と独立ＶＤM記述の形式化に対する対応表より仕様書と独立ＶＤM記述の変換規則を保証した。この問題点については、これは提案手法を使用せずに仕様書から独立ＶＤM記述を獲得する際にも生じる。本研究の独立ＶＤM記述から全体の振る舞いに変換する為の仕様書の情報整理からは逸脱している問題点であるが完璧な形式化を目指すうえでは変換規則の定式化の必要性がある。

仕様書を形式化する為の情報がお互い参照しているという問題点は、仕様書を章ごとに分割して他章の情報を参照しない様に形式化を行った。また、他章に関するデータ型を引用型を用いて抽象化することより１対１の仕様書の形式化が可能となった。これにより、もとあるドキュメントから仕様書の形式化が可能となった。

独立ＶＤM記述の一貫性は明確に確認しずらかったが統合ＶＤM記述の問題点の独立ＶＤM記述と統合ＶＤM記述の一貫性の解決方法を適用によって明示的に確認できるようになった。

よって本研究で提案した手法による仕様書の形式化は有効である。

謝辞

北陸先端科学技術大学院大学安心電子研究センター青木利晃准教授には本研究を進めるあたり親切なご指導を頂き、厚く御礼申し上げるとともに深謝の意を表します。また有益な助言をいただいた同大学院矢竹健朗助教に深謝いたします。また本研究を行う上で相談に乗っていただいた青木研究室土肥雅俊さん、谷崎裕明さん、Chaiwat Sathawornwichit さんに感謝の意を表します。また有意義な研究生活を提供してくださった北陸先端科学技術大学院大学に感謝したします。

参考文献

[1] 佐原伸, 形式手法の技術講座, 株式会社ソフト・リサーチセンター,2008 年

[2] プログラム仕様記述論, 荒木啓二郎 張漢明, 株式会社オーム社,2002 年

[3] http://www.vdmtools.jp/modules/tinyd1/index.php?id=1

[4] http://portal.osekvdx.org/index.php?option=com_frontpage&Itemid=1

第7章　付録

7.1　対応関係表

7.2　OSEK/VDXの独立VDM記述

```
class ActivateTask

types public
TaskID ::ID : <TaskIdentifies>
  T_State : State;

types public
State = <Ready>
|<Suspended>;


types public
Event = <event>

types public
Event_Set = set of Event

operations public
Activate_Task: TaskID ==>()
Activate_Task(TaskID) == return ;

/*Syntax: StatusType ActivateTask ( TaskType <TaskID> )
Parameter (In):
TaskID Task reference
Parameter (Out): none*/

operations public
```

要求仕様書（本文）		独立 VDM 記述
Syntax: StatusType ActivateTask (TaskType <TaskID>) Parameter (In):TaskID Task reference Parameter (Out): none		operations public Activate_Task: TaskID ==> () Activate_Task(TaskID) == return
Description: The task <TaskID> is transferred from the suspended state into the ready state14. The operating system ensures that the task code is b eing executed from the first statement. ” ”		operations public ActivateTask_Description: TaskID ==> TaskID ActivateTask_Description (mk_TaskID (<TaskIdentifies>T_sbjT_State)) == return mk_TaskID (<TaskIdentifies>,T_sbj,<Ready>) pre T_State = <Suspended> post T_State = <Ready>
If E_OS_LIMIT is returned the activation is ignored.		operations public ActivateTask_Particularites2: Status ==> () ActivateTask_Particularites2(Status) == if(Status = <E_OS_LIMIT>) then return <ignore_activation>
When an extended task is transferred from suspended state into ready state all its events are cleared		operations public ActivateTask_Particularites3: TaskID*Event_Set ==> Event_Set ActivateTask_Particularites3 (mk_TaskID (<TaskIdentifies>T_sbjT_State) Event_Set) == if(T_sbj=<ExtendedTask>) then return {} else return undefined
Status: Standard: ? No error E_OK ? Too many task activations of <TaskID> E_OS_LIMIT Extended: ? Task <TaskID> is invalid E_OS_ID		operations public ActivateTask_Status: Status_Case ==> Status ActivateTask_Status(Status_Case) == cases Status_Case: <NoError> -> return <E_OK> <ManyActivationsOfTaskID> -> return <E_OS_LIMIT> <TaskIDIsInvalid> -> return <E_OS_ID> end
Syntax: StatusType ChainTask (TaskType <TaskID>) Parameter (In): TaskID Reference to the sequential succeeding task to be activated. Parameter (Out): none		operations public Chain_Task:TaskID ==> () Chain_Task(TaskID) == return
Description: This service causes the termination of the calling task.		operations public ChainTaskDescription1:Task ==> Task ChainTaskDescription1(mk_Task(IDT_StateT_resourece)) == return <Tarmination>mk_Task(IDT_resourece)
After termination of the calling task a succeeding task <TaskID> is activated.		operations public ChainTaskDescription2:Task ==> Task ChainTaskDescription2(mk_Task(IDT_StateT_resourece)) == return mk_Task(ID<Activate>T_resourece)
”Using this service, it ensures that the succeeding task ” starts to run at the earliest ”after the calling task has been terminated. ” ”		operations public ChainTaskDescription3:Task ==> Task ChainTaskDescription3(mk_Task(ID,T_State,T_resourece)) == return mk_Task(ID,<run>,T_resourece)
” ”The task is not transferred to the suspended state, ” but will immediately become ready again. ” ” ”		operations public ChainTaskParticularities:Task ==> Task ChainTaskParticularities(mk_Task(ID,T_State,T_resourece)) == return mk_Task(ID,<ready>,T_resourece)
” An internal resource assigned to the calling task is ”automatically released, even if the succeeding task is i ” ”dentical with the current task. ” ” ”		operations public ChainTaskParticularities2:Task ==> Task ChainTaskParticularities2(mk_Task(ID,T_State,T_resource)) == return mk_Task(ID,T_State,T_resource) pre T_resource = <assigned> post T_resource = <released>
Other resources occupied by the calling shall have been released before ChainTask is called. ” ” ” ”		operations public ChainTaskParticularities3:Task*SystemCall ==> Task ChainTaskParticularities3 (mk_Task(ID,T_State,T_resource),ChainTask) == return mk_Task(ID,T_State,T_resource) pre T_resource = <released>
If a resource is still occupied in standard status the behaviour is undefined. ” ”		operations public ChainTaskParticularities7:Task*SystemCall ==> Task ChainTaskParticularities7 (mk_Task(ID,T_State,T_resource),ChainTask) == return undefined pre T_resource = <released>
”If called successfully, ChainTask does not return to ” the call level and the status can not be evaluated. In case of error the service returns to the calling task and provides a status which can then be evaluated in the application.		operations public ChainTaskParticularities4:SystemCall ==> () ChainTaskParticularities4(ChainTask) == return ;
”If the service ChainTask is called successfully, ” this enforces a rescheduling.		operations public ChainTaskParticularities5:SystemCall ==> rescheduling ChainTaskParticularities5(ChainTask) == return <rescheduling>
Ending a task function without call to TerminateTask or ChainTask is strictly forbidden and may leave the system in an undefined state. ” ”		operations public ChainTaskParticularities6:SystemCall*Task ==> Task ChainTaskParticularities6 (systemCall , mk_Task(ID,T_State,T_resource)) == return undefined pre systemCall = (<TerminateTask> or <ChainTask>)

表 7.1: 対応関係表

Status: Standard: ? No return to call level "? Too many task activations of <TaskID>, E_OS_LIMIT " "Extended: ? Task <TaskID> is invalid, E_OS_ID " "? Calling task still occupies resources, E_OS_RESOURCE " "? Call at interrupt level, E_OS_CALLEVEL " " " " " "	types public Status = <E_OS_LIMIT> —<E_OS_ID> —<E_OS_RESOURCE> —<E_OS_CALLLEVEL> types public StatusCase = <Too_many_task_activations> —<Task_is_invalid> —<Calling_task_still_occupies_resources> —<Call_at_interrupt_level> operations public Status_Case:StatusCase ==> Status Status_Case(StatusCase) == cases StatusCase: <Too_many_task_activations> -> return <E_OS_LIMIT>, <Task_is_invalid> -> return <E_OS_ID>, <Calling_task_still_occupies_resources> -> return <E_OS_RESOURCE>, <Call_at_interrupt_level> -> return <E_OS_CALLLEVEL> end
Syntax: DeclareResource (<ResourceIdentifier>) Parameter (In): ResourceIdentifier Resource identifier (C-identifier)	types public ResourceIdentifier = <ResourceIdentifier>; operations public Declare_Resource:ResourceIdentifier ==> () Declare_Resource(ResourceIDentifier) == return ;
Description: DeclareResource serves as an external declaration of a resource. The function and use of this service are similar to that of the external declaration of variables. " "	types public Resource = <Resource> types public ResoureceSet = set of Resource operations public DeclareResoureceDescription: Resource*ResoureceSet ==> ResoureceSet DeclareResoureceDescription(Resource,ResoureceSet) == return ResoureceSet union {Resource}
Parameter (In): TaskIdentifier Task identifier (C-identifier)	types public TaskIdentifier = <Empty>—<C_identifier>
Syntax: DeclareTask (<TaskIdentifier>) Parameter (In): TaskIdentifier Task identifier (C-identifier)	operations public DeclareTask: TaskIdentifier ==> () DeclareTask(TaskIdentifier) == return ;
the external declaration of variables	types public TaskIDset : set of TaskIdentifier ;
Description: DeclareTask serves as an external declaration of a task. The function and use of this service	operations public DeclareTask_Discription: TaskIdentifier ==> TaskIDset DeclareTask_Discription(TaskIdentifier) == return TaskIDset union {TaskIdentifier}
13.4.3.1 GetResource Syntax: StatusType GetResource (ResourceType <ResID>) Parameter (In): ResID Reference to resource Parameter (Out): none	types public ResID = <ResID>; operations public Get_Resource: ResID ==> () Get_Resource(ResID) == return ;
Particularities: The OSEK priority ceiling protocol for resource management is described in chapter 8.5.	/*undef*/
Nested resource occupation is only allowed if the inner critical sections are completely executed within the surrounding critical section " (strictly stacked, see chapter 8.2, " Restrictions when using resources).	/*undef*/
Nested occupation of one and the same resource is also forbidden!	types public Resource = <Get> —<Free> operations public GetResoureceDescription:Resource ==> Resource GetResoureceDescription(Resource) == return <Get> pre Resource = <Free>

表 7.2: 対応関係表

/* It is recommended that corresponding calls to GetResource and ReleaseResource appear within the same function. / /* "It is not allowed to use services which are points of " "rescheduling for non preemptable tasks " "(TerminateTask,ChainTask, Schedule and WaitEvent, " see chapter 4.6.2) in critical "sections. Additionally, critical sections are to be left before " completion of an interrupt service routine. "Generally speaking, critical sections should be short. " The service may be called from an ISR and from task level (see Figure 12-1). /	types public SystemCall = <GetResource> —<ReleaseResource> operations public GetResoureceDescription2:(Resource*SystemCall) ==> GetResoureceDescription2(mk_(Resource,SystemCall)) == return mk_(Resource,<ReleaseResource>) pre SystemCall = <GetResource> post SystemCall = <ReleaseResource>
Status: "Standard: ? No error, E_OK " "Extended: ? Resource <ResID> is invalid, E_OS_ID " "? Attempt to get a resource which is already occupied " "by any task or ISR, or the statically assigned priority of " "the calling task or interrupt routine is higher than the " " calculated ceiling priority, E_OS_ACCESS " " "	operations public GetResourceStatus:Status_Case ==> Status GetResourceStatus(Status_Case) == cases Status_Case: <NoError> -> return <E_OK>, <Invalid> -> return <E_OS_ID>, <Resouce_occupied> -> return <E_OS_ACCESS>, <Priority_Calling_Error> -> return <E_OS_ACCESS>, <interrupt_routine_higher_ceiling_priority> -> return <E_OS_ACCESS>, others -> return undefined end;
Parameter (In): none Parameter (Out): none	operations public Schedule:() ==> () Schedule() == return;
"if a higher-priority task is ready, the internal resource " "of the task is released, the current task is put into the " " ready state, its context is saved and the higher-priority task " "is executed. Otherwise the calling task is continued. " " " " " " "	operations public Schedule_Description1: Task*Task ==> (Task*Task) Schedule_Description1(mk_Task (T_Priority1,T_State1,T_Resource1,T_ContextState1), mk_Task(T_Priority2,T_State2,T_Resource2,T_ContextState2)) == return mk_(mk_Task (T_Priority1,<Running>,<NULL>,T_ContextState1) , mk_Task(T_Priority2,<Ready>,T_Resource2,<Save>)) pre T_State1 = <Ready> and T_State2 = <Running> and T_Priority1 > T_Priority2
Status: "Standard: ? No error, E_OK " "Extended: ? Call at interrupt level, E_OS_CALLEVEL " "? Calling task occupies resources, E_OS_RESOURCE " " " " " " "	types public Status = <E_OK> —<E_OS_CALLEVEL> —<E_OS_RESOURCE> types public Status_Case = <No_Error> —<Call_At_Intterrupt_Level> —<Calling_Task_Occupies_Resource> operations public Schedule_Status: Status_Case ==> Status Schedule_Status(Status_Case) == cases Status_Case: <No_Error> -> return <E_OK>, <Call_At_Intterrupt_Level> -> return <E_OS_CALLEVEL>, <Calling_Task_Occupies_Resource> -> return <E_OS_RESOURCE>, others -> return undefined end
The value 0 is defined as the lowest priority of a task. Accordingly bigger numbers define higher priorities.	types public Task = <task>; types public the_basis_of_the_task_priority = nat;
The scheduler decides on the basis of the task priority (precedence) which is the next of the ready tasks to be transferred into the running state.	operations public scheduler: the_next_of_the_ready_tasks ==> the_basis_of_the_task_priority * Task scheduler(the_basis_of_the_tasks_priority) == max_priority_task(the_basis_of_the_tasks_priority)

表 7.3: 対応関係表

A task being released from the waiting state is treated like
the last (newest) task in the ready queue of its priority
	types public the_ready_queue_of_its_priority :: priority :nat
	ready_queue :seq of Task;
	
	operations
	public task_being_released_from_the_waiting_state :
	the_ready_queue_of_its_priority ==> Task
	task_being_released_from_the_waiting_state
	(the_ready_queue_of_its_priority) ==
	newest_task(the_ready_queue_of_its_priority.ready_queue)
	functions
	public newest_task : seq of Task -> Task
	
	newest_task(ready_queue) == if
	(tl(ready_queue) = [])
	then
	hd(ready_queue)
	else
	newest_task(tl(ready_queue));

Several tasks of different priorities are in the ready state;
"i.e. three tasks of priority 3, one of priority 2 and one of
" " priority 1, plus two tasks of priority 0.
" "
"
"
"
"
	values public priority3 =
	mk_the_ready_queue_of_its_priority(3,[<task>,<task>,<task>]);
	values public priority2 =
	mk_the_ready_queue_of_its_priority(2,[<task>]);
	values public priority1 =
	mk_the_ready_queue_of_its_priority(1,[<task>]);
	values public priority0 =
	mk_the_ready_queue_of_its_priority(1,[<task>,<task>]);

" The scheduler selects the next task to be processed " (priority 3, first queue). Priority 2 tasks can only be " processed after all tasks of higher priority shall have left " the running and ready state, i.e. " " " " " " " " "	functions public usecase_Scheduler:() -> seq of bool usecase_Scheduler() == [t1]; static public t1: () -> bool t1() == let x = example_Task, scheduler2(x) = mk_the_ready_queue_of_its_priority(a,b), c = hd b in mk_the_ready_queue_of_its_priority(a, c) = mk_the_ready_queue_of_its_priority(3,[<task>]) pre The_processor = <terminated_a_task>;
To avoid the problems of priority inversion and deadlocks the OSEK operating system requires following behaviour "	functions public z:EachResource*CeilingPriority -> set of (<Resource>*nat) z(A,B) == forall { (a,b)—a in set A , b in set B}
" "? At the system generation, to each resource its own " ceiling priority is statically assigned. " " "	types public Resource :: CeilingPriority : nat ResoureceInf : <ResInf>; operations public SystemGeneration: Resource*nat ==> Resourece SystemGeneration(Resource,x) == return mk_Resourece(x,<ResInf>)
" The ceiling priority shall be set at least to the highest priority of all tasks that access a resource or any of the resources linked to this resource. The ceiling priority shall be lower than the lowest priority of "all tasks that do not access the resource, and which have " "priorities higher than the highest priority " of all tasks that access the resource.	types public Task :: priority :nat TaskInf :<TaskInf> operations public AccessResource:Task*Resource ==> () AccessResource(Task,Resource) == return undefined pre Task.priority < Resource.CeilingPriority
Syntax: StatusType ReleaseResource (ResourceType <ResID>) Parameter (In): ResID Reference to resource Parameter (Out): none	types public ResID = <ResID>; operations public Release_Resource: ResID ==> () Release_Resource(ResID) == return ;
Description: ReleaseResource is the counterpart of GetResource and serves to leave critical sections in the code that are assigned to the resource referenced by <ResID>. "Particularities: For information on nesting conditions, see " particularities of GetResource. The service may be called from an ISR and from task level (see Figure 12-1).	/*undef*/

表 7.4: 対応関係表

Status: ”Standard: ? No error, E_OK ” ”Extended: ? Resource <ResID> is invalid, E_OS_ID ” ? Attempt to release a resource which is not occupied by ” any task or ISR, or another resource shall be released ” ”before, E_OS_NOFUNC ” ? Attempt to release a resource which has a lower ceiling priority than the statically assigned priority of the calling ”task or interrupt routine, E_OS_ACCESS ” ”Conformance: BCC1, BCC2, ECC1, ECC2 ” ” ” ” ” ” ”	types public Status_Case = <NoError> —<ResID_invalid> —<Not_Occupied_Resource> —<Release_Resource_Lower_Ceiling_Priority> operations public ReleaseResourceStatus: Status_Case ==> Status ReleaseResourceStatus(Status_Case) == cases Status_Case: <NoError> -> return <E_OK>, <ResID_invalid> -> return <E_OS_ID>, <Not_Occupied_Resource> -> return <E_OS_NOFUNC>, <Release_Resource_Lower_Ceiling_Priority> -> return <E_OS_ACCESS>, others -> return undefined end;
Syntax: StatusType TerminateTask (void) Parameter (In): none Parameter (Out): none	operations public Terminate_Task:() ==> () Terminate_Task() == return
Description: This service causes the termination of the calling task. The calling task is transferred from the running state into the suspended state15. ” ” ” ”	types public TaskID ::ID : <TaskIdentifies> T_sbj : Task_Subject T_State : State; types public State = <Running> —<Suspended>; types public Task_Subject = <BasicTask> —<ExtendedTask> types public InternalResource = <Resource> —<None> operations public TerminateTask_Description: TaskID ==> TaskID TerminateTask_Description (mk_TaskID (<TaskIdentifies>,T_Subject,T_State)) == return mk_TaskID (<TaskIdentifies>,T_Subject,<Suspended>) pre T_State = <Running> post T_State = <Suspended>
An internal resource assigned to the calling task is automatically released. Other resources occupied by the task shall have been released before the call to ”TerminateTask. ”	operations public TerminateTask_Particularities1:((TaskID*InternalResource)) ==> InternalResource TerminateTask_Particularities1(mk_(TaskID,InternalResource)) == return <None>
If a resource is still occupied in standard status the behaviour is undefined. ” ”	types public Resource = InternalResource —<OtherResource> types public Call_TerminateTask_Timing = <yet> —<expiration> operations public TerminateTask_Particularities2: Resource*Call_TerminateTask_Timing ==> () TerminateTask_Particularities2 (Resource,Call_TerminateTask_Timing) == if (Resource=<OtherResource>) then return undefined pre Call_TerminateTask_Timing = <expiration>
”If the call was successful, TerminateTask does not return ” to the call level and the status can not be evaluated. ”If the version with extended status is used, the service ” ”returns in case of error, and provides a status which can be ” evaluated in the application. ”If the service TerminateTask is called successfully, ” it enforces a rescheduling.	/*undef*/
Ending a task function without call to TerminateTask or ChainTask is strictly forbidden and may leave the system in an undefined state. ” ” ” ” ” ” ” ”	operations public TerminateTask_Particularities3: TaskID*SystemServices ==> TaskID TerminateTask_Particularities3 (mk_TaskID (<TaskIdentifies>,T_Subject,T_State) ,SystemServices) == cases SystemServices: <TerminateTask> -> return mk_TaskID (<TaskIdentifies>,T_Subject,<Suspended>), <ChainTask> -> return mk_TaskID (<TaskIdentifies>,T_Subject,<Suspended>), others -> return undefined end post T_State = <Suspended>

表 7.5: 対応関係表

Status: Standard: No return to call level ”Extended: ? Task still occupies resources, E_OS_RESOURCE ” ”? Call at interrupt level, E_OS_CALLEVEL ” ”Conformance: BCC1, BCC2, ECC1, ECC2 ” ” ”	types public Status = <E_OS_RESOURCE> —<E_OS_CALLEVEL> types public Status_Case = <Task_still_occupies_resorces> —<Call_at_interrupt_level> operations public TerminateTask_Status: Status_Case ==> Status TerminateTask_Status(Status_Case) == cases Status_Case: <Task_still_occupies_resorces> -> return <E_OS_RESOURCE>, <Call_at_interrupt_level> -> return <E_OS_CALLEVEL> end
A new task is set into the ready state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction	types public NewTask = <Suspended>;

表 7.6: 対応関係表

```
ActivateTask_Description: TaskID ==>TaskID
ActivateTask_Description( mk_TaskID ( <TaskIdentifies>,T_State) )
== return mk_TaskID (<TaskIdentifies>,<Ready>)
pre T_State = <Suspended>
post T_State = <Ready>

/*Description: The task <TaskID> is transferred from the suspended state
into the ready state14.
 The operating system ensures that the task code is being executed from
 the first statement.*/

types public
TaskState = <ignore_activation>

operations public
ActivateTask_Particularites2: Status ==>TaskState
ActivateTask_Particularites2(Status) == return <ignore_activation>
pre Status = <E_OS_LIMIT>


/*If E_OS_LIMIT is returned the activation is ignored.*/
/*operations public
ActivateTask_Particularites3:TaskID*Event_Set ==>Event_Set
ActivateTask_Particularites3( mk_TaskID ( <TaskIdentifies>,T_State), Event_Set)
==
then return  {}
else return undefined;
```

図より	/*状態*/ /*起動*/ operations public Activate: State ==> State Activate(T_State) == cases T_State: <Suspended> -> return <Ready> end pre T_State = <Suspended> /*post T_State = <Ready>; */ /*開始*/ operations public Start: State ==> State Start(T_State) == cases T_State: <Ready> -> return <Running> end pre T_State = <Ready> /*post T_State = <Running>; */ /*待ち*/ operations public Preempt: State ==> State Preempt(T_State) == cases T_State: <Running> -> return <Ready> end pre T_State = <Running> /*post T_State = <Ready>; */ /*解放*/ operations public Terminate: State ==> State Terminate(T_State) == cases T_State: <Running> -> return <Suspended> end pre T_State = <Running> /*post T_State = <Suspended>; */ operations public Basic_Task : Transition * State ==> State
” ” ” ” ” ” ” ”	Basic_Task(S_Call,T_State) == cases S_Call: <Activate> -> Activate(T_State), <Start> -> Start(T_State), <Preempt> -> Preempt(T_State), <Terminate> -> Terminate(T_State) end;

表 7.7: 対応関係表

”System services are called from tasks, interrupt service ” ”routines, hook routines, and alarm-callbacks ” ” ”	types public CallSystemCallObject = <interrupt_service_routines>, —<hook_routines>, —<alarm__callbacks> types public SystemCall = <SystemCall> operations public Call_SystemCall:CallSystemCallObject ==> SystemCall Call_SystemCall(CSC) == return <SystemCall>;
12 Behaviour of system services is only defined for the services marked in the table.	/*undef*/
13 It may happen that currently no task is running. In this case the service returns the task ID INVALID_TASK (see chapter 13.2.3.5 GetTaskID).	types public NoRunningTask = <Task> types public ServiceReturn = <INVALID_TASK> operations public TerminateTask:NoRunningTask ==> ServiceReturn TerminateTask(NoRunningTask) == return <INVALID_TASK>
Task activation is performed using the operating system services ActivateTask or ChainTask	types public OS_Services = <Activate_Task> —<Chain_Task>;
The OSEK operating system does not support C-like parameter passing when starting a task. Those parameters should be passed by message ”communication (see chapter 10, Messages) ” or by global variables.	public Parameters = <Message_Communication> —<Global_Variables>; /*neg C-like Parameter_Passing*/
After activation the task is ready to execute from the first statement.	public Task_Config = <First_Statement> —<Others_Statement>; public Activate_Task_Config_State = Task_Config; public State = <Running> —<Ready> —<Suspended> —<Waiting>; operations public Activation_Task : OS_Services ==> State Activation_Task(OS_Services) == is not yet specified; /*post Activate_Task_Config_State = <First_Stattement>;*/

Multiple requesting of task activation means that the OSEK
operating system receives and records parallel
activations of a basic task already activated.
	types Task = State;
	types Activat_Task = Task;
	types OS_Rec = set of Activat_Task;

表 7.8: 対応関係表

```
*/
/*When an extended task is transferred from suspended state into
ready state all its events are cleared.*/

types public
Status = <E_OK>
|<E_OS_LIMIT>
|<E_OS_ID>

types public
Status_Case = <NoError>
|<ManyActivationsOfTaskID>
|<TaskIDIsInvalid>

operations public
ActivateTask_Status: Status_Case ==>Status
ActivateTask_Status(Status_Case) == cases Status_Case:
<NoError>->return <E_OK>,
<ManyActivationsOfTaskID>->return <E_OS_LIMIT>,
<TaskIDIsInvalid>->return <E_OS_ID>
end

/*
Status:
Standard: ? No error, E_OK
? Too many task activations of <TaskID>, E_OS_LIMIT
Extended: ? Task <TaskID> is invalid, E_OS_ID
*/

end ActivateTask

class Activating_A_Task
/*4.3*/
/*%%%%%%%%%%%%%%%%%%
Activating a task
%%%%%%%%%%%%%%%%%%%*/
types
public OS_Services  =  <Activate_Task>
|<Chain_Task>;
```

```
/*Task activation is performed using the operating system services
ActivateTask or ChainTask*/


public Parameters = <Message_Communication>
|<Global_Variables>; /*neg C-like Parameter_Passing*/
/*The OSEK operating system does not support C-like parameter passing
when starting a task.
Those parameters should be passed by message communication
(see chapter 10, Messages) or by global variables.*/


public Task_Config = <First_Statement>
|<Others_Statement>;

public Activate_Task_Config_State = Task_Config;

public State = <Running>
|<Ready>
|<Suspended>
|<Waiting>;

operations

public Activation_Task :  OS_Services ==> State
Activation_Task(OS_Services) == is not yet specified;

/*post
Activate_Task_Config_State = <First_Stattement>;*/

/*After activation the task is ready to execute from the first statement.*/

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Multiple requesting of task activation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

types Task = State;
types Activat_Task = Task;
```

```
types  OS_Rec = set of Activat_Task;

/*"Multiple requesting of task activation" means that the OSEK operating system
receives and records parallel activations of a basic task already activated.*/


types public Request = Task*State;

values public The_Maximum_Number_Of_Multiple_Requests
= undefined;/*未定義*/

types  public Request_Queued = seq of Request
inv
Request_Queued_len ==
len (Request_Queued_len)<=The_Maximum_Number_Of_Multiple_Requests;

types public Requests_of_Basic_Task_Activations :: priority:undefined
Request_Q:seq of Request

operations public
Request_queued : Requests_of_Basic_Task_Activations*Request* priority
=> Requests_of_Basic_Task_Activations
Request_queued(Queued,mk_(mk_(Task*State),priority)) ==
if(Request.priority = priority)
then
return Request.Request_Q^Request
else
return undefined

pre State = <Ready>

/*The number of multiple requests in parallel is defined in
a basic task specific attribute during system generation.
 If the maximum number of multiple requests has not been reached,
 the request is queued. The requests of basic task activations are
 queued per priority in activation order.*/
```

```
end Activating_A_Task

class BasicTask

/*4 タスク管理*/

types  /*4.1*/
/*Terminate;   下記で定義
Chenge_Higher_Prority_Task;
Interrupt_Service_Routine;
Release;
Prosesser;*/


/*4.2.2*/
public Transition  =  <Activate>
|<Start>
|<Preempt>
|<Terminate>;

public State = <Running>
|<Ready>
|<Suspended>;

public T_State = State;
public S_Call = Transition;

/*public Description =  Ready_By_System_Service
|Ready_By_Scheduler
|Scheduler_Decides_Start_Another_Task;
*/
public Basic_Task_S = State;

types public
NewTask = <Suspended>;

/*A new task is set into the ready state by a system service.
The OSEK operating system ensures that the execution of
the task will start with the first instruction*/
```

```
/*状態*/
/*起動*/
operations
public Activate: State ==> State

Activate(T_State)  == cases T_State:
<Suspended>->return <Ready>
end

pre
T_State  = <Suspended>

/*post
T_State = <Ready>;
*/

/*開始*/
operations
public Start:  State ==> State

Start(T_State) == cases T_State:
<Ready>->return <Running>
end

pre
T_State = <Ready>

/*post
T_State = <Running>;
*/

/*待ち*/
operations
public Preempt: State ==> State

Preempt(T_State) == cases T_State:
<Running>->return <Ready>
end
```

```
pre
T_State = <Running>
/*post
T_State = <Ready>;
*/

/*解放*/
operations
public Terminate: State ==> State

Terminate(T_State) == cases T_State:
<Running>->return <Suspended>
end

pre
T_State = <Running>
/*post
T_State = <Suspended>;
*/
operations
public Basic_Task : Transition  * State  ==> State

Basic_Task(S_Call,T_State)  ==  cases S_Call:
<Activate>  ->  Activate(T_State),
<Start>->Start(T_State),
<Preempt>->Preempt(T_State),
<Terminate>->Terminate(T_State)
end;


end BasicTask

class ChainTask

types public
TaskID = <TaskID>
operations public
Chain_Task:TaskID ==>()
```

```
Chain_Task(TaskID) == return

/*
Syntax: StatusType ChainTask ( TaskType <TaskID> )
Parameter (In):
TaskID Reference to the sequential succeeding task to be activated.
Parameter (Out): none
*/

types public
State = <Activate>
|<Tarmination>
|<run>
|<ready>

types public
Task :: ID :<TaskID>
T_State :State
T_resourece :Resourece



operations public
ChainTaskDescription1:Task ==>Task
ChainTaskDescription1(mk_Task(ID,T_State,T_resourece)) ==
return mk_Task(ID,<Tarmination>,T_resourece)

/*Description: This service causes the termination of the calling task.*/

operations public
ChainTaskDescription2:Task ==>Task
ChainTaskDescription2(mk_Task(ID,T_State,T_resourece)) ==
return mk_Task(ID,<Activate>,T_resourece)


/* After termination of the calling task a succeeding task <TaskID> is activated. */

operations public
```

```
ChainTaskDescription3:Task ==>Task
ChainTaskDescription3(mk_Task(ID,T_State,T_resourece)) ==
return mk_Task(ID,<run>,T_resourece)

/*Using this service, it ensures that the succeeding task starts to run at the earlie

types public
requests = <request>
/*undef*/
/*Particularities: If the succeeding task is identical with the current task,
 this does not result in multiple requests.*/
operations public
ChainTaskParticularities:Task ==>Task
ChainTaskParticularities( mk_Task(ID,T_State,T_resourece) ) ==
return mk_Task(ID,<ready>,T_resourece)

/* The task is not transferred to the suspended state,
 but will immediately become ready again.*/

types public
Resourece =  <assigned>
|<released>

operations public
ChainTaskParticularities2:Task ==>Task
ChainTaskParticularities2(mk_Task(ID,T_State,T_resource)) ==
return mk_Task(ID,T_State,T_resource)
pre T_resource = <assigned>
post T_resource = <released>

/*
An internal resource assigned to the calling task is automatically released,
 even if the succeeding task is identical with the current task.*/

operations public
ChainTaskParticularities3:Task*SystemCall ==>Task
ChainTaskParticularities3(mk_Task(ID,T_State,T_resource),ChainTask) ==
return mk_Task(ID,T_State,T_resource)
```

```
pre T_resource  =  <released>
/* Other resources occupied by the calling shall have been released
before ChainTask is called. */

operations public
ChainTaskParticularities7:Task*SystemCall ==>Task
ChainTaskParticularities7(mk_Task(ID,T_State,T_resource),ChainTask) ==
return undefined
pre T_resource  =  <released>

/*If a resource is still occupied in standard status the behaviour
is undefined.*/

operations public
ChainTaskParticularities4:SystemCall ==>()
ChainTaskParticularities4(ChainTask) == return ;

/*If called successfully, ChainTask does not return to
the call level and the status can not be evaluated.
In case of error the service returns to the calling task and
provides a status which can then be evaluated in the application.*/

types public
rescheduling = <rescheduling>

operations public
ChainTaskParticularities5:SystemCall ==>rescheduling
ChainTaskParticularities5(ChainTask) == return  <rescheduling>

/*
If the service ChainTask is called successfully,
this enforces a rescheduling.
*/


types public
SystemCall = <TerminateTask>
|<ChainTask>
```

```
|<OteherCall>

operations public
ChainTaskParticularities6:SystemCall*Task ==>Task
ChainTaskParticularities6( systemCall , mk_Task(ID,T_State,T_resource) )
== return undefined
/*pre systemCall =  ( <TerminateTask> or <ChainTask> )*/

/*
Ending a task function without call to TerminateTask or
ChainTask is strictly forbidden and may leave the system in an undefined state.*/

/*If E_OS_LIMIT is returned the activation is ignored.
When an extended task is transferred from suspended state into ready state all its ev
*/

types public
Status = <E_OS_LIMIT>
|<E_OS_ID>
|<E_OS_RESOURCE>
|<E_OS_CALLLEVEL>
types public
StatusCase = <Too_many_task_activations>
|<Task_is_invalid>
|<Calling_task_still_occupies_resources>
|<Call_at_interrupt_level>

operations public
Status_Case:StatusCase ==>Status
Status_Case(StatusCase) == cases StatusCase:
<Too_many_task_activations>->return <E_OS_LIMIT>,
<Task_is_invalid>->return <E_OS_ID>,
<Calling_task_still_occupies_resources>->
return <E_OS_RESOURCE>,
<Call_at_interrupt_level>->return <E_OS_CALLLEVEL>
end

/*Status:
```

```
Standard: ? No return to call level
? Too many task activations of <TaskID>, E_OS_LIMIT
Extended: ? Task <TaskID> is invalid, E_OS_ID
? Calling task still occupies resources, E_OS_RESOURCE
? Call at interrupt level, E_OS_CALLEVEL*/

end ChainTask

class Conformance_classes

types public

ConformanceClass = <BCC1>
|<BCC2>
|<ECC1>
|<ECC2>

types public
Multiple_Requesting_of_Task_Activation = <BCC2>
|<ECC2>


values public
Number_of_Tasks_Which_are_not_in_the_SuspnedeState_BCC = 8
values public
Number_of_Tasks_Which_are_not_in_the_SuspnedeState_ECC = 16
/*any combinations of BT&ET*/
types public
More_Then_One_Task_Pre_Priority = <BB2>
|<ECC2>
/*ECC2 is both BT&ET*/
types public
Resoruce :: BCC1:<RES_SCHEDULER>
BCC2:nat
ECC1:nat
ECC2:nat

values public
resoruce = mk_Resoruce(<RES_SCHEDULER>,8,8,8)
```

```
values public
Internal_Resources = 2
values public
Alarm = 1
values public
ApplicationMode = 1

end Conformance_classes

class DeclareResource

types public
ResourceIdentifier = <ResourceIdentifier>;

operations public
Declare_Resource:ResourceIdentifier ==> ()
Declare_Resource(ResourceIDentifier) == return ;

/*Syntax: DeclareResource ( <ResourceIdentifier> )
Parameter (In):
ResourceIdentifier Resource identifier (C-identifier)*/

types public
Resource = <Resource>

types public
ResoureceSet = set of Resource


operations public
DeclareResoureceDescription: Resource*ResoureceSet ==>ResoureceSet
DeclareResoureceDescription(Resource,ResoureceSet) ==
return ResoureceSet union {Resource}
/*Description: DeclareResource serves as an external declaration of
a re-source.
The function and use of this service are similar to that
of the external declaration of variables.*/
```

```
end DeclareResource


class Declare_Task


types public
TaskIdentifier = <Empty>|<C_identifier>


/*Parameter (In):
TaskIdentifier Task identifier (C-identifier)*/

types public
TaskIDset : set of TaskIdentifier ;

/*the external declaration of variables*/

/*operations public
DeclareTask: TaskIdentifier ==>()
DeclareTask(TaskIdentifier) == return ;
*/

/*Syntax: DeclareTask ( <TaskIdentifier> )
Parameter (In):
TaskIdentifier Task identifier (C-identifier)*/

operations public
DeclareTask_Discription: TaskIdentifier ==>TaskIDset
DeclareTask_Discription(TaskIdentifier) ==
return TaskIDset union {TaskIdentifier}

/*Syntax: DeclareTask ( <TaskIdentifier> )*/
/*Description: DeclareTask serves as an external declaration of a task.
The function and use of this service are similar to that of
the external declaration of variables.*/
```

```
end Declare_Task


class ExtendedTask
/*4.2.1*/

types
public Transition  =  <Activate>
|<Start>
|<Preempt>
|<Terminate>
|<Wait>
|<Release>;

public State = <Running>
|<Ready>
|<Suspended>
|<Waiting>;

public E_State = State;


operations
public Activate: State ==>State

Activate(E_State) == cases E_State:
<Suspended>->return <Ready>

end
pre
E_State = <Suspended>
/*post
Activate(E_State) = <Ready>;*/

operations
public Start: State ==>State

Start(E_State) == cases E_State:
<Ready>->return <Running>
```

```
end
pre
E_State = <Ready>
/*post
Start(E_State) = <Running>;*/

operations
public Wait: State ==>State

Wait(E_State) == cases E_State:
<Running>->return <Waiting>
end
pre
E_State = <Running>
/*post
Wait(E_State) = <Waiting>;*/


operations
public Release : State ==>  State

Release(E_State) == cases E_State:
<Waiting>->return <Ready>
end
pre
E_State = <Waiting>
/*post
Release(E_State) = <Ready>;*/

operations
public Preempt : State ==>State

Preempt(E_State) == cases E_State:
<Running>->return <Ready>
end
pre
E_State = <Running>
/*post
```

```
Preempt(E_State) = <Ready>;
*/
operations
public Terminate : State ==>State

Terminate(E_State) == cases E_State:
<Running>->return <Suspended>
end
pre
E_State = <Running>
/*post
Terminate(E_State) = <Suspended>;
*/
operations
public Extended_Task : Transition * State ==>State

Extended_Task(S_Call , E_State) == cases S_Call:
<Activate>->Activate(E_State),
<Start>->Start(E_State),
<Wait>->Wait(E_State),
<Release>->Release(E_State),
<Preempt>->Preempt(E_State),
<Terminate>->Terminate(E_State)
end;


end ExtendedTask

class GetResource

types public
ResID = <ResID>;

operations public
Get_Resource: ResID ==> ()
Get_Resource(ResID) == return ;

/*13.4.3.1 GetResource
Syntax: StatusType GetResource ( ResourceType <ResID> )
```

```
Parameter (In):
ResID Reference to resource
Parameter (Out): none*/

/*undef*/
/*
Description: This call serves to enter critical sections in the code that
are assigned to the resource referenced by <ResID>.
 A critical section shall always be left using ReleaseResource.
Particularities: The OSEK priority ceiling protocol for
resource management is described in chapter 8.5.
*/


/*
Nested resource occupation is only allowed if the inner critical sections
are completely executed within the surrounding critical
section (strictly stacked, see chapter 8.2, Restrictions when using resources).
*/
types public
Resource =  <Get>
|<Free>
operations public
GetResoureceDescription:Resource ==> Resource
GetResoureceDescription(Resource) == return <Get>
pre Resource = <Free>


/*
Nested occupation of one and the same resource is also forbidden!
*/
types public
SystemCall = <GetResource>
|<ReleaseResource>

operations public
GetResoureceDescription2:(Resource*SystemCall) ==> (Resource*SystemCall)
GetResoureceDescription2(mk_(Resource,SystemCall)) ==
```

```
return mk_(Resource,<ReleaseResource>)
pre SystemCall = <GetResource>
post SystemCall = <ReleaseResource>


/*
It is recommended that corresponding calls to GetResource and
ReleaseResource appear within the same function.
*/

/*
It is not allowed to use services which are points of rescheduling
for non preemptable tasks (TerminateTask,ChainTask, Schedule and WaitEvent,
 see chapter 4.6.2) in critical sections. Additionally, critical sections
 are to be left before completion of an interrupt service routine.
Generally speaking, critical sections should be short.
The service may be called from an ISR and from task level (see Figure 12-1).
*/
types public
Status = <E_OK>
|<E_OS_ID>
|<E_OS_ACCESS>;
types public
Status_Case = <NoError>
|<Invalid>
|<Resouce_occupied>
|<Priority_Calling_Error>
|<interrupt_routine_higher_ceiling_priority>


operations public
GetResourceStatus:Status_Case ==>Status
GetResourceStatus(Status_Case) == cases Status_Case:
<NoError>->return <E_OK>,
<Invalid>->return <E_OS_ID>,
<Resouce_occupied>->return <E_OS_ACCESS>,
<Priority_Calling_Error>->return <E_OS_ACCESS>,
<interrupt_routine_higher_ceiling_priority>->
```

```
return <E_OS_ACCESS>,
others ->return undefined
end;
/*
Status:
Standard: ? No error, E_OK
Extended: ? Resource <ResID> is invalid, E_OS_ID
? Attempt to get a resource which is already occupied by any task or ISR,
 or the statically assigned priority of the calling task
 or interrupt routine is higher than the calculated ceiling priority,
  E_OS_ACCESS
Conformance: BCC1, BCC2, ECC1, ECC2*/

end GetResource


class InternalResource

types public
system_functions = <GetResource>
|<ReleaseResource>
|<InternallyManaged>

operations public
InternalResoureceManaged:system_functions ==>()
InternalResoureceManaged(system_functions) == return undefined

pre system_functions = <InternallyManaged>

/*Internal resources are resources which are not visible to the user
and therefor can not be addressed by the system functions GetResource
and ReleaseResource. Instead, they are managed strictly internally
within a clearly defined set of system functions. */


/*Besides that, the behaviour of internal resources is exactly
the same as standard resources (priority ceiling protocol etc.).*/
```

```
/*うまくかけない*/

/*operations public
system_generation:set of task ==>set of (task*nat)
system_generation(t) == let
x = {},
k = x munion {mk_(forall a in set t , 1)}
in
return k*/

/*Internal resources are restricted to tasks. At most one internal
resource can be assigned to a task during system generation.*/

types public
State = <Running>

types public
task :: T_State : State
T_res : Resource;

types public
Resource = <Resource>
|<none>

operations public
InaternalResource: task*Resource ==>task
InaternalResource (mk_task(T_State,T_res),Resource) ==
return mk_task(<Running>,<Resource>)
pre
T_res  = <none>
post
T_State = <Running> and T_res = <Resource>

/*The resource is automatically taken when the task enters the running state9,
 except when it has already taken the resource. As a result, the priority
 of the task is automatically changed to the ceiling priority
 of the resource.*/
```

```
end InternalResource


class Legitimacy_of_calls

types public
CallSystemCallObject = <interrupt_service_routines>,
|<hook_routines>,
|<alarm__callbacks>
types public
SystemCall = <SystemCall>

operations public
Call_SystemCall:CallSystemCallObject ==>SystemCall
Call_SystemCall(CSC) == return <SystemCall>;

/*System services are called from tasks, interrupt service routines,
 hook routines, and alarm-callbacks.*/


/*undef*/
/*12 Behaviour of system services is only defined for the services marked
in the table.*/

types public
NoRunningTask = <Task>

types public
ServiceReturn = <INVALID_TASK>

operations public
TerminateTask:NoRunningTask ==>ServiceReturn
TerminateTask(NoRunningTask) == return <INVALID_TASK>

/*13 It may happen that currently no task is running.
In this case the service returns the task ID INVALID_TASK
(see chapter 13.2.3.5 GetTaskID).*/
```

```
end Legitimacy_of_calls

class OSEKPriorityCeilingProtocol

types public
EachResource = set of <Resource>

types public
CeilingPriority = nat

types public
Ceilling = undefined

types public
Resource_P = <Resource>*CeilingPriority

functions public
z:EachResource*CeilingPriority ->set of (<Resource>*nat)
z(A,B) ==  forall { (a,b)|a in set A , b in set B}



/*To avoid the problems of priority inversion and deadlocks
the OSEK operating system requires following behaviour*/
types public
Resource :: CeilingPriority  : nat
    ResoureceInf : <ResInf>;

operations public
SystemGeneration: Resource*nat ==>Resourece
SystemGeneration(Resource,x) == return mk_Resourece(x,<ResInf>)
/*? At the system generation, to each resource its own ceiling priority is
statically assigned.*/

types public
Task :: priority :nat
TaskInf :<TaskInf>
operations public
AccessResource:Task*Resource ==>()
```

```
AccessResource(Task,Resource) == return undefined
pre Task.priority < Resource.CeilingPriority

/* The ceiling priority shall be set at least to the highest priority of
all tasks that access a resource or
 any of the resources linked to this resource.
The ceiling priority shall be lower than the lowest priority of all tasks
that do not access the resource,
 and which have priorities higher than the highest priority of
 all tasks that access the resource.

/*
? If a task requires a resource, and its current priority
is lower than the ceiling priority of the resource,

the priority of the task is raised to the ceiling priority of the resource.
? If the task releases the resource, the priority of this task is reset to the priori


end OSEKPriorityCeilingProtocol


class ReleaseResource

types public
ResID = <ResID>;

operations public
Release_Resource: ResID ==>()
Release_Resource(ResID) == return ;

/*Syntax: StatusType ReleaseResource ( ResourceType <ResID> )
Parameter (In):
ResID Reference to resource
Parameter (Out): none
*/

/*undef*/
/*
```

```
Description: ReleaseResource is the counterpart of GetResource and serves
to leave critical sections in the code that are assigned
to the resource referenced by <ResID>.
Particularities: For information on nesting conditions,
see particularities of GetResource.
The service may be called from an ISR and from task level (see Figure 12-1).
*/

types public
Status = <E_OK>
|<E_OS_ID>
|<E_OS_NOFUNC>
|<E_OS_ACCESS>;

types public
Status_Case = <NoError>
|<ResID_invalid>
|<Not_Occupied_Resource>
|<Release_Resource_Lower_Ceiling_Priority>

operations public
ReleaseResourceStatus: Status_Case ==>Status
ReleaseResourceStatus(Status_Case) == cases Status_Case:
<NoError>-> return <E_OK>,
<ResID_invalid>-> return  <E_OS_ID>,
<Not_Occupied_Resource>-> return <E_OS_NOFUNC>,
<Release_Resource_Lower_Ceiling_Priority>->
return  <E_OS_ACCESS>,
others -> return undefined
end;
/*
Status:
Standard: ? No error, E_OK
Extended: ? Resource <ResID> is invalid, E_OS_ID
? Attempt to release a resource which is not occupied by any task or ISR,
 or another resource shall be released before, E_OS_NOFUNC
? Attempt to release a resource which has a lower ceiling priority than
the statically assigned priority of the calling task or
```

```
interrupt routine, E_OS_ACCESS
Conformance: BCC1, BCC2, ECC1, ECC2*/

end ReleaseResource


class schedule

types public
State = <Ready>
  |<Running>

types public
Resourece = <NULL>
    |<GET>

types public
Task :: T_Priority :nat
T_State  :State
T_Resource : Resourece
T_ContextState : ContextState

types public
ContextState = <Save>
|<Oteher>

types public
Taskset = set of Task

operations public
Schedule:() ==> ()
Schedule() == return;

/*Parameter (In): none
Parameter (Out): none*/

operations public
Schedule_Description1: Task*Task ==> (Task*Task)
Schedule_Description1
```

```
(mk_Task(T_Priority1,T_State1,T_Resource1,T_ContextState1),
mk_Task(T_Priority2,T_State2,T_Resource2,T_ContextState2)) ==
return mk_( mk_Task(T_Priority1,<Running>,<NULL>,T_ContextState1) ,
 mk_Task(T_Priority2,<Ready>,T_Resource2,<Save>) )

pre T_State1 = <Ready> and T_State2 = <Running> and
T_Priority1 > T_Priority2

/*if a higher-priority task is ready, the internal resource of the task is
released, the current task is put into the ready state, its context is
saved and the higher-priority task is executed. Otherwise the calling task is
 continued.*/

/*Rescheduling only takes place if the task an internal resource is
assigned to the calling task*/

/*Schedule enables a processor assignment to other tasks with lower or
equal priority than the ceiling priority of the internal resource and
higher priority than the priority of the calling task in
application-specific locations. When returning from Schedule,
 the internal resource has been taken again.
This service has no influence on tasks
with no internal resource assigned (preemptable tasks).*/

types public
Status =  <E_OK>
|<E_OS_CALLEVEL>
|<E_OS_RESOURCE>
types public
Status_Case =  <No_Error>
|<Call_At_Intterrupt_Level>
|<Calling_Task_Occupies_Resource>

operations public
Schedule_Status: Status_Case ==>Status
Schedule_Status(Status_Case) == cases  Status_Case:
<No_Error>->return <E_OK>,
<Call_At_Intterrupt_Level>->return <E_OS_CALLEVEL>,
```

```
<Calling_Task_Occupies_Resource>->return <E_OS_RESOURCE>,others ->  return undefined
end

/*Status:
Standard: ? No error, E_OK
Extended: ? Call at interrupt level, E_OS_CALLEVEL
? Calling task occupies resources, E_OS_RESOURCE*/

end schedule


class Task_Priority

types public Task = <task>;
types public the_basis_of_the_task_priority = nat;

/*The value 0 is defined as the lowest priority of a task.
Accordingly bigger numbers define higher priorities.*/

types public the_next_of_the_ready_tasks =
set of (the_basis_of_the_task_priority * Task);

operations
public scheduler: the_next_of_the_ready_tasks ==>
the_basis_of_the_task_priority * Task

scheduler(the_basis_of_the_tasks_priority) ==
max_priority_task(the_basis_of_the_tasks_priority)

/*The scheduler decides on the basis of the task priority (precedence)
which is the next of the ready tasks to be transferred into the running state.*/

/*ready タスクの集合から優先度の高い物をポップする補助関数*/
functions
public max_priority_task : set of (nat * <task>) ->nat * <task>

 max_priority_task(z) ==
   let
mk_(x,c1) in set z
```

```
be st
    forall mk_(y,c2) in set z \ {mk_(x,c1)} & x > y
in mk_(x,c1)
pre
z <>{};


functions
public f: set of nat -> set of (nat*nat)
f(t) =
let s
be st
 forall mk(x,1) in set s & exists x in t

in s

/* print a.max_priority_task({mk_(1,<task>),mk_(2,<task>)})*/


/*どっちがより厳密か？*/
types public the_next_of_the_ready_tasks_2 :: priority :nat
ready_list :seq of Task;

/*どっちがより厳密か？ 一回エラーがでると動く ••• */

operations
public scheduler2: set of the_next_of_the_ready_tasks_2 ==>
the_next_of_the_ready_tasks_2
scheduler2(the_basis_of_the_tasks_priority) ==
return max_priority_task2(the_basis_of_the_tasks_priority);


functions
public max_priority_task2 : set of the_next_of_the_ready_tasks_2 ->
the_next_of_the_ready_tasks_2

 max_priority_task2(z) ==
   let
```

```
mk_the_next_of_the_ready_tasks_2(x,c1) in set z
be st
    forall mk_the_next_of_the_ready_tasks_2(y,c2) in set z \
    {mk_the_next_of_the_ready_tasks_2(x,c1)} & x > y
in mk_the_next_of_the_ready_tasks_2(x,c1)
pre
z <>{};


/*print a.max_priority_task2
({mk_the_next_of_the_ready_tasks_2(1,[<task>,<task>]),
mk_ the_next_of_the_ready_tasks_2(2,[<task>])} ) */

/*print a.max_priority_task2
({mk_Task_Priority`the_next_of_the_ready_tasks_2(1,[<task>,<task>]),
mk_Task_Priority`the_next_of_the_ready_tasks_2(2,[<task>])})*/



/*print a.max_priority_task1
( { mk_the_basis_of_the_tasks_priority2(1,[<task>,<task>]),
mk_the_basis_of_the_tasks_priority2(2,[<task>,<task>]),
mk_the_basis_of_the_tasks_priority2(3,[<task>,<task>]) } )*/

types
public State = <Running>
|<Ready>
|<Suspended>
|<Waiting>;

instance variables
private 『Task_State』 : State := <Ready>;

/*
The scheduler decides on the basis of the task priority (precedence)
which is the next of the ready tasks to be transferred into the running state.
The value 0 is defined as the lowest priority of a task.
Accordingly bigger numbers define higher priorities.
```

```
*/
/*undef*/
/*To enhance efficiency, a dynamic priority management is not supported.*/

operations
public started_depending: () ==>Task

started_depending() == return undefined

/*Tasks on the same priority level are started depending on their order of activation
 whereby extended tasks in the waiting state do not block the start of
  subsequent tasks of identical priority.*/


types public ready_list = seq of Task


types public ready_list_of_its_current_priority :: priority :nat
ready_list :seq of Task;

operations
public preempted_task: ready_list_of_its_current_priority ==>Task
preempted_task(ready_list_of_its_current_priority) ==
oldest_task_in_the_ready_list
(ready_list_of_its_current_priority.ready_list);

/*リストの先頭を返す補助関数*/
functions
public oldest_task_in_the_ready_list : seq of Task ->Task
oldest_task_in_the_ready_list(ready_list) == hd(ready_list);

/*A preempted task is considered to be the first (oldest) task
in the ready list of its current priority.*/

types public the_ready_queue_of_its_priority :: priority :nat
ready_queue :seq of Task;

operations
```

```
public task_being_released_from_the_waiting_state :
the_ready_queue_of_its_priority ==>Task

task_being_released_from_the_waiting_state(the_ready_queue_of_its_priority)
== newest_task(the_ready_queue_of_its_priority.ready_queue)


functions
public newest_task : seq of Task ->Task

newest_task(ready_queue) == if
(tl(ready_queue) = [])
then
hd(ready_queue)
else
newest_task(tl(ready_queue));


/*A task being released from the waiting state is treated like the last
 (newest) task in the ready queue of its priority*/

operations public
remove_task:the_ready_queue_of_its_priority ==> the_ready_queue_of_its_priority
remove_task(readyqueue) == is not yet specified

pre len readyqueue > len redyqueque~

/*<usecase>*/

/*Figure 4-5 shows an example implementation of the scheduler using
for each priority level.*/

values public  priority3 =
mk_the_ready_queue_of_its_priority(3,[<task>,<task>,<task>]);
values public  priority2 =
mk_the_ready_queue_of_its_priority(2,[<task>]);
values public priority1 =
mk_the_ready_queue_of_its_priority(1,[<task>]);
```

```
values public priority0 =
mk_the_ready_queue_of_its_priority(1,[<task>,<task>]);



/*Several tasks of different priorities are in the ready state;
i.e. three tasks of priority 3, one of priority 2 and one of priority 1,
 plus two tasks of priority 0.*/

/*<undef>*/

/*The task which has waited the longest time, depending on its
order of requesting, is shown at the bottom of each queue.
Figure 4-5 shows an example implementation of the scheduler using
for each priority level.The processor has just processed
and terminated a task*/

values public example_Task = {priority3,priority2,priority1,priority0};

values public The_processor = <terminated_a_task>;

/*
functions
public usecase_Scheduler:() -> seq of bool

usecase_Scheduler() == [t1];



static public t1: () -> bool
t1() == let
x = example_Task,
scheduler2(x) = mk_the_ready_queue_of_its_priority(a,b),
c = hd b
in
mk_the_ready_queue_of_its_priority(a, c) = mk_the_ready_queue_of_its_priority(3,[<tas

pre The_processor = <terminated_a_task>;
```

```
*/
/*The scheduler selects the next task to be processed (priority 3, first queue).
Priority 2 tasks can only be processed after all tasks of higher priority
 shall have left the running and ready state, i.e.*/

/*動作テスト*/
/*functions
 public minimum : set of (nat*<task>) -> (nat*<task>)
 minimum(z) ==
   let
mk_(x,c1) in set z
be st
    forall mk_(y,c2) in set z \ {mk_(x,c1)} & x < y
in mk_(x,c1)
pre
 z <> {}




types public test0 :: a:nat
b:seq of Task;

functions public test4: test0 ->test0

test4(x) == x;

functions public test5: set of test0 ->set of test0
test5(x) == x;


functions
public test: the_basis_of_the_tasks_priority2 ->
the_basis_of_the_tasks_priority2
test(x)== x;
```

```
 print a.test(mk_the_basis_of_the_tasks_priority2(1,[<task>]))


functions
public test2: set of the_basis_of_the_tasks_priority2 ->
set of the_basis_of_the_tasks_priority2
test2(x) == x;

*/
end Task_Priority



class TerminateTask

operations public
Terminate_Task:() ==>()
Terminate_Task() == return

/*Syntax: StatusType TerminateTask ( void )
Parameter (In): none
Parameter (Out): none*/

types public
TaskID ::ID : <TaskIdentifies>
  T_sbj : Task_Subject
  T_State : State;
types public
State = <Running>
|<Suspended>;

types public
Task_Subject = <BasicTask>
|<ExtendedTask>

types public
InternalResource =  <Resource>
|<None>
```

```
operations public
TerminateTask_Description: TaskID ==>TaskID
TerminateTask_Description( mk_TaskID (<TaskIdentifies>,T_Subject,T_State))
==
return mk_TaskID (<TaskIdentifies>,T_Subject,<Suspended>)

pre  T_State = <Running>
post T_State = <Suspended>

/*Description: This service causes the termination of the calling task.
The calling task is transferred from the running state
into the suspended state15.*/

operations public
TerminateTask_Particularities1:((TaskID*InternalResource)) ==>
InternalResource
TerminateTask_Particularities1( mk_(TaskID,InternalResource) ) ==
return <None>

/*An internal resource assigned to the calling task is automatically released.
Other resources occupied by the task shall have been released
before the call to TerminateTask.*/

types public
Resource = InternalResource
  |<OtherResource>
types public
Call_TerminateTask_Timing = <yet>
|<expiration>

operations public
TerminateTask_Particularities2:Resource*Call_TerminateTask_Timing ==>()
TerminateTask_Particularities2(Resource,Call_TerminateTask_Timing) ==
if (Resource=<OtherResource>)
then return undefined
pre Call_TerminateTask_Timing = <expiration>

/*If a resource is still occupied in standard status the behaviour
```

```
is undefined.*/

types public
SystemServices = <TerminateTask>
|<ChainTask>;

/*undef*/
/*If the call was successful, TerminateTask does not return to
the call level and the status can not be evaluated.
If the version with extended status is used, the service returns
in case of error, and provides a status which can be evaluated
in the application.
If the service TerminateTask is called successfully, it enforces
 a rescheduling.*/

operations public
TerminateTask_Particularities3:TaskID*SystemServices ==>TaskID
TerminateTask_Particularities3( mk_TaskID (<TaskIdentifies>,T_Subject,T_State),
SystemServices) ==
cases SystemServices:
<TerminateTask>->return mk_TaskID (<TaskIdentifies>,T_Subject,<Suspended>),
<ChainTask>->return mk_TaskID (<TaskIdentifies>,T_Subject,<Suspended>),
others ->return undefined
end

post T_State = <Suspended>


/*Ending a task function without call to TerminateTask or ChainTask is
strictly forbidden and may leave the system in an undefined state.*/

types public
Status = <E_OS_RESOURCE>
|<E_OS_CALLEVEL>

types public
Status_Case = <Task_still_occupies_resorces>
|<Call_at_interrupt_level>
```

```
operations public
TerminateTask_Status: Status_Case ==>Status
TerminateTask_Status(Status_Case) == cases Status_Case:
<Task_still_occupies_resorces>->
return <E_OS_RESOURCE>,
<Call_at_interrupt_level>->return <E_OS_CALLEVEL>
end

/*Status:
Standard: No return to call level
Extended: ? Task still occupies resources, E_OS_RESOURCE
? Call at interrupt level, E_OS_CALLEVEL
Conformance: BCC1, BCC2, ECC1, ECC2*/

end TerminateTask
```

7.3 OSEK/VDXの統合VDM記述

```
/*
注意：
これのサービスコールは一部を除きすべてタスクレベル(実行しているタスクでの関数呼
び出し)で呼び起されています。(ISRは未実装)
Declaer～はアプリケーションで作られているタスクをOSに認識させる関数です。
従ってDeclear～はタスクレベルで呼ばれる物ではありません。
初期のタスクはアプリケーションが自動的に最初実行するタスクを決めます。
また、一回目スケジューラで起動したタスクが自動的に実行するタスクであると仮定しま
す。


*/

class impvdm
```

```
types public
State = <Running>
|<Ready>
|<Suspended>
|<Waiting>;



types public
Task_Subject = <BasicTask>
|<ExtendedTask>

types public
Event = <event>

types public
Event_Set = set of Event

types public
Status = <E_OK>
|<E_OS_LIMIT>
|<E_OS_ID>
|<E_OS_ACCESS>
|<E_OS_CALLEVEL>
|<E_OS_RESOURCE>
|<E_OS_NOFUNC>
|<INVALID_TASK>


types public
Status_Case = <NoError>
|<ManyActivationsOfTaskID>
|<TaskIDIsInvalid>
|<Invalid>
|<Resouce_occupied>
|<Priority_Calling_Error>
```

```
|<interrupt_routine_higher_ceiling_priority>
|<Call_At_Intterrupt_Level>
|<Calling_Task_Occupies_Resource>
|<Task_still_occupies_resorces>
|<Call_at_interrupt_level>

types
public OS_Services  =  <Activate_Task>
|<Chain_Task>;

types
public Parameters = <Message_Communication>
|<Global_Variables>; /*neg C-like Parameter_Passing*/

types
public Task_Config = <First_Statement>
|<Others_Statement>;

types
public Activate_Task_Config_State = Task_Config;

types Task1 = State;
types Activat_Task1 = Task1;
types  OS_Rec = set of Activat_Task1;

/*"Multiple requesting of task activation" means that the OSEK
operating system receives and records parallel activations
of a basic task already activated.*/


types public Request = <ActivateTask>;

values public The_Maximum_Number_Of_Multiple_Requests = 3/*undefined;*/


types  public Request_Queued = seq of Request
inv
Request_Queued_len ==
```

```
len (Request_Queued_len)<=The_Maximum_Number_Of_Multiple_Requests;

types public Requests_of_Basic_Task_Activations :: BasicTask:<BasicTask>
Request_Q:seq of Request

types public
ResID = <ResID>;

types public
ResourceState = <Get>
|<Free>;

types public
SystemCall = <GetResource>
|<ReleaseResource>;


types public
TaskIdentifier = <Empty>|<C_identifier>


/*Parameter (In):
TaskIdentifier Task identifier (C-identifier)*/

types public
TaskIDset = set of TaskIdentifier ;

types public
system_functions = <GetResource>
|<ReleaseResource>
|<InternallyManaged>

types public
ContextState = <Save>
|<Oteher>

types public
Taskset = set of Task
```

```
types public
InternalResource =  <Resource>
|<None>



types public
Call_TerminateTask_Timing = <yet>
|<expiration>


types public
SystemServices = <TerminateTask>
|<ChainTask>;



types public the_basis_of_the_task_priority = nat;
types public the_next_of_the_ready_tasks =
set of (the_basis_of_the_task_priority * Task);

types public the_next_of_the_ready_tasks_2 :: priority :nat
ready_list :seq of Task;




types public ready_list = seq of Task


types public ready_list_of_its_current_priority :: priority :nat
ready_list :seq of Task;


types public the_ready_queue_of_its_priority :: priority :nat
ready_queue :seq of Task;
```

```
types public
Task ::ID : nat
  T_sbj : Task_Subject
  T_State : State
  T_priroity : nat
  T_res : T_Resource
  T_ContextState : ContextState ;

types public
T_Resource =  seq of Resource

types public
TaskID = nat

types public
ResorceCnd = <Free>
|TaskID

types public
Resource :: ResorceID :nat
CeilingPriority :nat
ResorceCondetion:ResorceCnd

types public
ResoureceSet = set of Resource

types public
ResourceIdentifier = <ResourceIdentifier>;




instance variables
public 『Taskset』 : set of Task := {};

public 『RequestQueuedSet』: set of the_ready_queue_of_its_priority := { };
```

```
public 『Processor』 : set of Task := {};

public 『Status』 : set of Status := {};

public 『IDset』 : set of nat := {};

public 『ResourceSet』 : set of Resource := {};

public 『ResourceIDset』 : set of nat := {};



/*mk_impvdm‘Task(1,<BasicTask>,<Suspended>,1,[],<Oteher>)*/
/*mk_impvdm‘Resource(1,2,<Free>)*/
/*mk_impvdm‘Task(2,<BasicTask>,<Suspended>,5,
([mk_impvdm‘Resource(1,2,<Get>),mk_impvdm‘Resource(3,4,<Free>)]),<Oteher>)*/
/*[ mk_impvdm‘Resource(1,2,<Free>),mk_impvdm‘Resource(3,4,<Free>)
,mk_impvdm‘Resource(5,6,<Free>) ]*/
/*mk_impvdm‘the_ready_queue_of_its_priority
(2,[mk_impvdm‘Task(2,<BasicTask>,<Suspended>,5,([mk_impvdm‘Resource(1,2,<Get>),
mk_impvdm‘Resource(3,4,<Free>)]),<Oteher>),
mk_impvdm‘Task(3,<BasicTask>,<Suspended>,7,([mk_impvdm‘Resource(4,6,<Get>)
,mk_impvdm‘Resource(5,9,<Get>)]),<Oteher>)])*/

/*print a.ShiftSearchRequestQElementResFree
(mk_impvdm‘Task(2,<BasicTask>,<Suspended>,5,([mk_impvdm‘Resource(1,2,<Get>)
,mk_impvdm‘Resource(3,4,<Free>)]),<Oteher> ) , mk_impvdm‘Resource(3,4,<Free>)
, [mk_impvdm‘Task(2,<BasicTask>,<Suspended>,5,([mk_impvdm‘Resource(1,2,<Get>)
,mk_impvdm‘Resource(3,4,<Free>)]),<Oteher>),
mk_impvdm‘Task(3,<BasicTask>,<Suspended>,7,([mk_impvdm‘Resource(4,6,<Get>)
,mk_impvdm‘Resource(5,9,<Get>)]),<Oteher>)])*/

/*キューを生成*/
operations public
Declare_RequestQ: nat ==>set of the_ready_queue_of_its_priority
Declare_RequestQ(priority) == if (not(exists a in set 『RequestQueuedSet』 & a.priorit
then
```

```
return {mk_the_ready_queue_of_its_priority(priority,[])}
else
return {}

/*キューを追加*/

operations public
AddRequestQueueSet:
the_ready_queue_of_its_priority*(set of the_ready_queue_of_its_priority) ==>
set of the_ready_queue_of_its_priority
AddRequestQueueSet(RequestQueued,RequestQueuedSet) ==
return RequestQueuedSet union {RequestQueued}

/*pre RequestQueued not in set RequestQueuedSet
*/

/*タスク集合に追加*/
operations public
AddTaskset:Task*(set of Task) ==>set of Task
AddTaskset(mk_Task(ID,T_sbj,T_State,T_priority,T_res,T_ContextState),Taskset)
== return Taskset union
{mk_Task(ID,T_sbj,T_State,T_priority,T_res,T_ContextState)}

pre  mk_Task(ID,T_sbj,T_State,T_priority,T_res,T_ContextState) not in set Taskset;

/*ID の集合に追加*/
operations public
AddIDset:nat*set of nat ==>set of nat
AddIDset(ID,IDset) == return IDset union {ID}
pre ID not in set IDset;

/*Declare_Task*/
operations public
Declare_Task: Task ==>()
Declare_Task(mk_Task(ID,T_sbj,T_State,T_priority,T_res,T_ContextState)) ==      (
『RequestQueuedSet』 :=
『RequestQueuedSet』 union
 Declare_RequestQ(T_priority);
```

```
『Taskset』 :=
 AddTaskset
 (mk_Task(ID,T_sbj,T_State,T_priority,T_res,
 T_ContextState) , 『Taskset』);
『IDset』 := AddIDset(ID,『IDset』)
);
```

```
/*ActivateTask*/

/*Task集合からIDよりTaskを検索*/
operations public
TasksetIDSelection: nat ==> Task
TasksetIDSelection(ID) == let
a in set 『Taskset』
  be st
 a.ID =ID
in
return a

/*キュー集合から指定した優先度のキューを選択*/
operations public
RequestQueuedSetSelection:nat ==>the_ready_queue_of_its_priority
RequestQueuedSetSelection(priority) == let
a in set 『RequestQueuedSet』
be st
 a.priority = priority
in
return a
```

```
/*キュー集合から指定したキューを取り除く*/
operations public
SubRequestQueueSetElement:
set of the_ready_queue_of_its_priority * the_ready_queue_of_its_priority ==>
set of the_ready_queue_of_its_priority
SubRequestQueueSetElement(RequestQueuedSet,RequestQueued) ==
return RequestQueuedSet \ {RequestQueued}

/*リクエストキューにタスクを付け加える */
operations public
AddShiftRequestQueueSetElement: the_ready_queue_of_its_priority*Task ==>
the_ready_queue_of_its_priority
AddShiftRequestQueueSetElement(RequestQueueSetElement,Task) == let
a = RequestQueueSetElement.priority,
b = RequestQueueSetElement.ready_queue,
c = b^ [Task]
in
return
mk_the_ready_queue_of_its_priority(a,c)

/*リクエストキューの集合の要素から先頭のタスクを取り除く*/
operations public
RemoveHeadShiftRequestQueueSetElement:the_ready_queue_of_its_priority ==>
 the_ready_queue_of_its_priority
RemoveHeadShiftRequestQueueSetElement(RequestQueueSetElement)  == let
a = RequestQueueSetElement.priority,
b = RequestQueueSetElement.ready_queue,
c = tl b
in
return mk_the_ready_queue_of_its_priority(a,c)

/*タスク集合から除外*/
operations public
SubTasksetElement:set of Task * Task ==> set of Task
SubTasksetElement(Taskset,Task) == return Taskset \ {Task}
```

```
operations public
Activate_Task: nat ==> ()
Activate_Task(ID) == if ((ID not in set 『IDset』) or
(not(TasksetIDSelection(ID).T_State = <Suspended>)))
then
『Status』 := {<E_OS_ID>}
else
let
mk_Task(b,c,d,e,f,g) = TasksetIDSelection(ID),
Task = mk_Task(b,c,d,e,f,g),
TaskCP = CeilingPriorityTask(Task),
h = RequestQueuedSetSelection(TaskCP),
i = SubRequestQueueSetElement(『RequestQueuedSet』 , h ),
j = AddShiftRequestQueueSetElement
(h,mk_Task(b,c,<Ready>,e,f,g)),
k = SubTasksetElement( 『Taskset』, mk_Task(b,c,d,e,f,g)),
l = mk_Task(b,c,<Ready>,e,f,g)
in
(『RequestQueuedSet』 := i union { j };
『Taskset』:= AddTaskset(l,k);
『Status』 := {<E_OK>};
)

/*scheduler*/

/*最も大きいリクエストキューを選択*/

types public
MaxPRQType = the_ready_queue_of_its_priority
|<NULL>


operations public
MaxPriorityRequestQueuedSelection: set of the_ready_queue_of_its_priority  ==>
MaxPRQType
MaxPriorityRequestQueuedSelection(RequestQueuedSet) ==
if(RequestQueuedSet = {})
then
```

```
return <NULL>
else
if(card(RequestQueuedSet) = 1)
then
let
{mk_the_ready_queue_of_its_priority
(z,zseq)} = RequestQueuedSet
in
return mk_the_ready_queue_of_its_priority
(z,zseq)
else
let
mk_the_ready_queue_of_its_priority(x,xseq)
in set RequestQueuedSet
be st
forall mk_the_ready_queue_of_its_priority(y,yseq)
 in set RequestQueuedSet \
 {mk_the_ready_queue_of_its_priority(x,xseq)} & x>y
in
return mk_the_ready_queue_of_its_priority(x,xseq)

/*空じゃない最も大きいリクエストキューを選択*/
operations public
NoEmptyMaxPriorityRequestQueuedSelection:
set of the_ready_queue_of_its_priority ==>MaxPRQType
NoEmptyMaxPriorityRequestQueuedSelection(RequestQueuedSet) == let
a = MaxPriorityRequestQueuedSelection
(RequestQueuedSet)
in
if (a = <NULL>)
then
return <NULL>
else
if( a.ready_queue = [])
then

NoEmptyMaxPriorityRequest
QueuedSelection(RequestQueuedSet \
```

```
 {a})
else
return a

/*リクエストキューからタスクを取り出す*/
operations public
PopNoEmptyMaxPriorityRequestQueued:the_ready_queue_of_its_priority ==>
Task
PopNoEmptyMaxPriorityRequestQueued(NoEmptyMaxPriorityRequestQueued) ==
let
a =
NoEmptyMaxPriorityRequestQueued.
ready_queue,
b = hd a
in
return b

/*リクエストキューからタスクを取り出した後のリクエストキュー*/
operations public
DwellPopNoEmptyMaxPriorityRequestQueued:the_ready_queue_of_its_priority ==>
seq of Task
DwellPopNoEmptyMaxPriorityRequestQueued(NoEmptyMaxPriorityRequestQueued) ==
let
a =  NoEmptyMaxPriorityRequestQueued
.ready_queue,
b = tl a
in
return b
/*リクエストキューの集合の要素を取り除く*/
operations public
RemoveElementRequestQueueSet:
(set of the_ready_queue_of_its_priority)*the_ready_queue_of_its_priority ==>
set of the_ready_queue_of_its_priority
RemoveElementRequestQueueSet(RequestQueueSet,RequestQueue) ==
return RequestQueueSet \ {RequestQueue}


/*Ready->Running*/
```

```
operations public
ReadyRunning:Task ==>Task
ReadyRunning(Task) == let
mk_Task(a,b,c,d,e,f) = Task
in
return mk_Task(a,b,<Running>,d,e,f)
pre Task.T_State = <Ready>


/*Runnning → Ready 状態に移行*/
operations public
RunningReady:Task ==>Task
RunningReady(Task) == let
mk_Task(a,b,c,d,e,f) = Task
in
return mk_Task(a,b,<Ready>,d,e,f)
pre Task.T_State = <Running>


/*Task を Taskset 集合から取り除く*/
operations public
RemoveTaskset: Task*set of Task ==> set of Task
RemoveTaskset(Task,Taskset) ==  return Taskset \ {Task}




operations public
Scheduler:() ==>()
Scheduler() == if(『Taskset』 = {})
then
(
『Status』  := {<E_OK>}; /*何もしない*/
)
else
if( 『Processor』 = {})
then
let
```

```
NEMPQ = NoEmptyMaxPriorityRequestQueuedSelection
( 『RequestQueuedSet』 )
in
if(NEMPQ = <NULL>)
then
(
『Status』  := {<E_OK>}; /*何もしない*/
)
else
let
TASK = PopNoEmptyMaxPriorityRequestQueued(NEMPQ),
STE = SubTasksetElement(『Taskset』,TASK),
RQSS = RequestQueuedSetSelection(CeilingPriorityTask
(TASK))
in
(
『Processor』 := { ReadyRunning(TASK) };
『Taskset』 := STE union { ReadyRunning(TASK) };
『RequestQueuedSet』 :=
RemoveElementRequestQueueSet(『RequestQueuedSet』 , RQSS ) union {RemoveHeadShiftReque
『Status』  := {<E_OK>};
)
else
let
PC_TASK in set 『Processor』
in
if(NoEmptyMaxPriorityRequestQueuedSelection
(『RequestQueuedSet』 ) = <NULL>)
then
(
『Status』  := {<E_OK>}; /*何もしない*/
)
else
if(not (PC_TASK.T_res = []))
then
『Status』  :=  {<E_OS_RESOURCE>}
else
if((CeilingPriorityTask(PC_TASK) <
```

```
NoEmptyMaxPriorityRequestQueuedSelection
(『RequestQueuedSet』 ).priority ))
then
let
NEMPRQS1 = NoEmptyMaxPriorityRequest
QueuedSelection( 『RequestQueuedSet』),
PNEMPRQ1 = PopNoEmptyMaxPriority
RequestQueued(NEMPRQS1),
RDYTASK = ReadyRunning(PNEMPRQ1),
RDYRT = RemoveTaskset(PNEMPRQ1 ,
  『Taskset』 ),

RUNTASK = RunningReady(PC_TASK),
RUNRT = RemoveTaskset(PC_TASK,『Taskset』),
PC_RQS = RequestQueuedSetSelection
(CeilingPriorityTask(PC_TASK)),
ASRQSE = AddShiftRequestQueueSetElement
(PC_RQS , RUNTASK),
PC_RERQS = RemoveElementRequestQueueSet
( 『RequestQueuedSet』 , PC_RQS),
RHSRQSE = RemoveHeadShiftRequestQueueSet
Element(NEMPRQS1),
RERQS = RemoveElementRequestQueueSet
(『RequestQueuedSet』, NEMPRQS1),
TasksetPro = RDYRT union {RDYTASK},
ProRun = RemoveTaskset(PC_TASK,TasksetPro),
Tasksetall = ProRun union {RUNTASK}

in
( 『Processor』  := {RDYTASK};
 『Taskset』 := Tasksetall;
 『RequestQueuedSet』 := PC_RERQS
union {ASRQSE};
 『RequestQueuedSet』 := RERQS
union {RHSRQSE};
 『Status』  := {<E_OK>};
)
else
```

```
(
『Processor』 := {PC_TASK};
『Status』   := {<E_OK>};
)

/*TerminateTask*/

/*Running->Suspended*/
operations public
RunningSuspended:Task ==>Task
RunningSuspended(Task) == let
mk_Task(a,b,c,d,e,f) = Task
in
return mk_Task(a,b,<Suspended>,d,e,f)
pre Task.T_State = <Running>


operations public
Terminate_Task:() ==> ()
Terminate_Task() ==  if( 『Processor』 = {})
then
『Status』 := { <INVALID_TASK> }
else
let
a in set 『Processor』
in
if(a.T_res = [])
then
(
『Processor』 := {};
『Taskset』 :=
SubTasksetElement( 『Taskset』 , a )
union { RunningSuspended(a) };
)

else
『Status』 := {<E_OS_RESOURCE>}
```

```
/*DeclareResource*/
/*リソース集合にリソース追加*/
operations public
AddResourceSet:set of Resource * Resource ==>set of Resource
AddResourceSet(ResourceSet,Resource) == return ResourceSet union {Resource}

/*リソースＩＤ集合にＩＤ追加*/
operations public
AddResourceIDset:set of nat * nat ==> set of nat
AddResourceIDset(ResourceIDset,ResoreceID) ==
return ResourceIDset union {ResoreceID}

/*DeclareResource*/
operations public
Declare_Resource: Resource ==>()
Declare_Resource( mk_Resource(ResorceID,CeilingPriority,ResorceCondetion) ) == (
『ResourceSet』 :=
AddResourceSet( 『ResourceSet』
, mk_Resource(ResorceID,CeilingPriority,
ResorceCondetion) );
『ResourceIDset』 :=
AddResourceIDset( 『ResourceIDset』 ,
 ResorceID);
『RequestQueuedSet』
:= 『RequestQueuedSet』
union  Declare_RequestQ(CeilingPriority);
)

/*ISR 未実装の為 processor にあるタスクのみ資源が獲得可能*/
/*GetResource*/

types public
ResourceIDselectionTypeResource = Resource
|bool

/*資源の集合から指定した ID の資源を抽出*/
```

```
operations public
ResourceSetIDSelection: nat ==> ResourceIDselectionTypeResource
ResourceSetIDSelection(ResID) == let
a in set 『ResourceSet』
  be st
a.ResorceID = ResID
in
return a

/*タスクにリソースを追加*/
operations public
GetResourceTask:Task*Resource ==>Task
GetResourceTask( mk_Task(ID,T_sbj,T_State,T_priority,T_res,T_ContextState) ,
 AddResource ) == if (T_res = [])
then
return mk_Task
(ID,T_sbj,T_State,
T_priority,
[AddResource],
T_ContextState)
else
return mk_Task
(ID,T_sbj,T_State,
T_priority,
(T_res^[AddResource]),
T_ContextState)

/*タスクがもつ資源の中で最も大きい優先度を計算*/
operations public
MaxResorecePriority: seq of Resource ==>nat
MaxResorecePriority(Res) == let
FRes  = hd Res
in
if(tl Res = [])
then
return FRes.CeilingPriority
else
```

```
let
TlRes = tl Res,
SRes  = hd (TlRes),
TLTLRes = tl (tl Res)
in
if (FRes.CeilingPriority >
SRes.CeilingPriority)
then
return MaxResorecePriority([FRes]^TLTLRes)
else
return MaxResorecePriority(TlRes)


/*天井優先度を計算*/
operations public
CeilingPriorityTask: Task ==>nat
CeilingPriorityTask(mk_Task(ID,T_sbj,T_State,T_priority,T_res,T_ContextState))
== if (T_res = [])
then
return T_priority
else
if(MaxResorecePriority(T_res)>
T_priority)
then
return
MaxResorecePriority(T_res)
else
return T_priority ;
/*資源の ID チェック*/
operations public
CheackResorceID: (set of nat) * nat ==>bool
CheackResorceID( ResourceIDset , ResID ) ==
return  ResID in set ResourceIDset;

/*<Free>のリソースにタスクＩＤを記録*/
operations public
FreeGet:Resource * TaskID ==>Resource
FreeGet(mk_Resource(ID,CP,FG),TaskID) ==
```

```
return mk_Resource(ID,CP,TaskID)
pre FG = <Free>

/*リソースの集合から取り除く*/
operations public
SubResourceSetElement:set of Resource * Resource ==> set of Resource
SubResourceSetElement(ResourceSet,Resource) == return ResourceSet \ {Resource}

/*GetResource*/
operations public
Get_Resource:nat ==>()
Get_Resource(ResorceID) ==
if(not (CheackResorceID( 『ResourceIDset』 , ResorceID)))
then
『Status』 := {<E_OS_ID>}
else
let
proc in set 『Processor』
in
let
Res  =  ResourceSetIDSelection(ResorceID)

in
if(not (Res.ResorceCondetion = <Free>))
then
『Status』 := {<E_OS_ACCESS>}
else
if (CeilingPriorityTask(proc) <
 Res.CeilingPriority)
then
let
ProcID = proc.ID,
GRes  = FreeGet(Res,ProcID),
GetRT  =
GetResourceTask(proc,GRes),

SubTs  =
SubTasksetElement(『Taskset』 ,
```

```
 proc),
TS = SubTs union {GetRT},
SRSE =
SubResourceSetElement
( 『ResourceSet』 , Res)
in
(
『Processor』 := {GetRT};
『Taskset』 := TS;
『Status』  := {<E_OK>};
『ResourceSet』 := SRSE
union {GRes}
)
else
『Status』  := {<E_OS_ACCESS>}

pre  not( 『Processor』 = {})



/*ReleaseResource*/
/*タスク ID からタスクの優先度と同じリクエストキューを選択*/
operations public
RequestQIDSelection:nat*(set of the_ready_queue_of_its_priority) ==>
the_ready_queue_of_its_priority
RequestQIDSelection(ID,RequestQSet) == let
Task = TasksetIDSelection(ID),
TaskCP = CeilingPriorityTask(Task)
in
return RequestQueuedSetSelection(TaskCP)

/*タスクのスタックから資源の解放*/
operations public
ShiftTaskResourceInf:T_Resource*Resource ==>T_Resource
ShiftTaskResourceInf(T_res,mk_Resource(ID,CP,Cnd)) == if (hd T_res =
mk_Resource(ID,CP,Cnd))
then
return tl T_res
```

```
else
return [hd T_res] ^
ShiftTaskResourceInf
(tl (T_res),mk_Resource(ID,CP,Cnd))

/*タスクに取られている指定した資源の解放*/
operations public
FreeResourceTask:Task*Resource ==>Task
FreeResourceTask(
mk_Task(ID,T_sbj,T_State,T_priority,T_res,T_ContextState),Resource) ==
return mk_Task(ID,T_sbj,T_State,T_priority,
(ShiftTaskResourceInf(T_res,Resource)),T_ContextState)

/*リクエストキューの列から指定したタスクから指定した資源を解放*/
operations public
ShiftSearchRequestQElementResFree:Task*Resource*seq of Task ==>
seq of Task
ShiftSearchRequestQElementResFree
(mk_Task(ID,T_sbj,T_State,T_priority,T_res,T_ContextState),Resource,RequestQ)
==
if(hd RequestQ = mk_Task
(ID,T_sbj,T_State,T_priority,T_res,T_ContextState))
then
return
[mk_Task
(ID,T_sbj,
T_State,
T_priority,
(ShiftTask
ResourceInf
(T_res,
Resource)),
T_ContextState)
]^
tl RequestQ
else
return
[hd RequestQ]
```

```
^ ShiftSearch
RequestQ
ElementResFree
(mk_Task(ID,
T_sbj,T_State,
T_priority,
T_res,
T_ContextState)
,Resource,
 tl (RequestQ))

/*リクエストキューから指定したタスクの指定した資源の解放*/
operations public
RemoveTaskResouresInf:Task*the_ready_queue_of_its_priority*Resource ==>
the_ready_queue_of_its_priority
RemoveTaskResouresInf(Task,RequestQ,Resource) == let
RQ = RequestQ.ready_queue,
PRQ = RequestQ.priority,
NRQ = ShiftSearchRequestQElementResFree
(Task,Resource,RQ)
in
return
mk_the_ready_queue_of_its_priority(PRQ,NRQ);

/*タスクの資源の列の最後の要素*/
operations public
SeqLastElement:seq of Resource ==>Resource
SeqLastElement(Seq) == if(tl Seq = [])
then
return hd Seq
else
return SeqLastElement( tl Seq);

/*タスクの資源の列の最後の要素*/
types public
T_resLastElement = Resource

operations public
```

```
TResLastElement:T_Resource ==>T_resLastElement
TResLastElement(T_res) == if(T_res = [hd T_res])
then
return hd T_res
else
SeqLastElement(T_res);

/*指定したリソースの解放*/
ShiftResourceFree : Resource ==>Resource
ShiftResourceFree(mk_Resource(ResorceID,CeilingPriority,ResorceCondetion))
==
return mk_Resource(ResorceID,CeilingPriority,<Free>)

/*Release_Resourece*/
operations public
Release_Resource:nat ==>()
Release_Resource(ResID) == if(not (CheackResorceID( 『ResourceIDset』 , ResID)))
then
『Status』 := {<E_OS_ID>}
else
let
proc in set 『Processor』
in
let
Res  =  ResourceSetIDSelection(ResID)
in
if(Res.ResorceCondetion = <Free>)
then
『Status』 := {<E_OS_NOFUNC>}
else
let
PCPT  =  CeilingPriorityTask(proc),
GRIDT = Res.ResorceCondetion,
GRT = TasksetIDSelection(GRIDT),
GRTP = CeilingPriorityTask(GRT)
in
if(PCPT < GRTP)
then
```

```
『Status』 := {<E_OS_ACCESS>}
else
let
NT = FreeResourceTask(GRT,Res),
STS = SubTasksetElement
(『Taskset』,GRT),
FR = ShiftResourceFree(Res),
SRS = SubResourceSetElement
( 『ResourceSet』, Res)
in
cases GRT.T_State:
<Ready>->let
RQ  =
RequestQIDSelection
(GRT.ID,
『RequestQueuedSet』 ),
SRQ  =
SubRequestQueueSetElement
(『RequestQueuedSet』,RQ),
NRQ =
RemoveTaskResouresInf
(GRT,RQ,Res)

in
(
『RequestQueuedSet』
:= SRQ union {NRQ};
『Taskset』 :=
 STS union {NT};
『ResourceSet』
 := SRS union {FR};
『Status』  := {<E_OK>};
),
<Running> ->(
『Processor』  := {NT};
『Taskset』 := STS union {NT};
『ResourceSet』 :=
 SRS union {FR};
```

```
『Status』 := {<E_OK>};
),

others ->(
『Taskset』 := STS union {NT};
『ResourceSet』 :=
SRS union {FR};
『Status』 := {<E_OK>};
)
end

pre not (CheackResorceID( 『ResourceIDset』 , ResID )) or
ChckResoureceCondetion(ResID) and (not( 『Processor』= {}))
/*解放しようとしている資源が有る　ならば　タスクが獲得している資源の後尾が解放
しようとしている資源と同じである*/

/*リソースの列の後尾の資源を指定*/
operations public
TlResourceSeq:T_Resource ==>Resource
TlResourceSeq(ResourceSeq) == if([(hd ResourceSeq)] = ResourceSeq)
then
return (hd ResourceSeq)
else
return TlResourceSeq((tl ResourceSeq))



/*資源ＩＤと資源ＩＤを獲得しているタスクの資源の後尾が同じであるかを真偽でかえ
す*/
operations public
ChckResoureceCondetion:nat ==>bool
ChckResoureceCondetion(ResID) == if(ResourceSetIDSelection1(ResID) = false)
then
return  false
else
let
R_Cdt = ResourceSetIDSelection(ResID).
ResorceCondetion
```

```
in
if(R_Cdt=<Free>)
then
return false
else
let
Task = TasksetIDSelection(R_Cdt),
TaskRS = Task.T_res,
Res = TlResourceSeq(TaskRS)
in
return Res = ResourceSetIDSelection(ResID)

/*資源のＩＤから資源集合にＩＤに該当する資源ある場合資源を返しない場合 false を
かえす*/
operations public
ResourceSetIDSelection1: nat ==> ResourceIDselectionTypeResource
ResourceSetIDSelection1(ResID) == let
b in set 『ResourceSet』
in
if (b.ResorceID = ResID)
then
let
a in set 『ResourceSet』
  be st
a.ResorceID = ResID
in
return a
else
return false

/*ChainTask*/
operations public
Chain_Task:nat ==>()
Chain_Task(ID) == (
Terminate_Task();
Activate_Task(ID)
)
pre  not( 『Processor』= {})
```

```
/*
cr a:= new impvdm()
print a.Activate_Task(1)
print a.Activate_Task(2)
print a.Scheduler()
print a.Declare_Resource(mk_impvdm‘Resource(1,3,<Free>))
print a.Declare_Resource(mk_impvdm‘Resource(1,4,<Free>))
print a.Declare_Task(mk_impvdm‘Task(1,<BasicTask>,<Suspended>,2,[],<Oteher>))
print a.Declare_Task(mk_impvdm‘Task(2,<BasicTask>,<Suspended>,3,[],<Oteher>))
print a.Get_Resource(1)
print a.Get_Resource(2)
print a.Release_Resource(1)
print a.Release_Resource(2)

*/

instance variables
『x』 : nat := 0;

operations public
run:() ==>()
run() == while 『x』 < 10 do
(  ||(
Activate_Task(1),
Activate_Task(2),
Scheduler(),
Declare_Resource(mk_impvdm‘Resource(1,3,<Free>)),
Declare_Resource(mk_impvdm‘Resource(1,4,<Free>)),
Declare_Task(mk_impvdm‘Task(1,<BasicTask>,<Suspended>,2,[],
<Oteher>)),
Declare_Task(mk_impvdm‘Task(2,<BasicTask>,<Suspended>,3,[],
<Oteher>)),
Get_Resource(1),
Get_Resource(2),
Release_Resource(1),
Release_Resource(2),
```

```
『x』 := 『x』 +1
);

)

/*初期タスク ID1 が起動*/

operations public
run1:() ==>()
run1() == (
Declare_Resource(mk_impvdm‘Resource(1,3,<Free>));
Declare_Resource(mk_impvdm‘Resource(1,4,<Free>));
Declare_Task(mk_impvdm‘Task
(1,<BasicTask>,<Suspended>,2,[],<Oteher>));
Declare_Task(mk_impvdm‘Task
(2,<BasicTask>,<Suspended>,3,[],<Oteher>));
Activate_Task(1);
Scheduler();


while 『x』 < 10 do(
||(if(not (CheackResorceID( 『ResourceIDset』 , 1 ))
or ChckResoureceCondetion(1) and  (not( 『Processor』 = {})))
then
||(
Activate_Task(1),
Activate_Task(2),
Scheduler(),
Get_Resource(1),
Get_Resource(2),
Release_Resource(1)
  )
else skip,
if((not (CheackResorceID( 『ResourceIDset』 , 2 ))
or ChckResoureceCondetion(2) and  (not( 『Processor』 = {}))))
then
||(
Activate_Task(1),
```

```
Activate_Task(2),
Scheduler(),
Get_Resource(1),
Get_Resource(2),
Release_Resource(2)
)
else skip,
|| (
Activate_Task(1),
Activate_Task(2),
Scheduler(),
Get_Resource(1),
Get_Resource(2),

));

『x』 := 『x』 +1;
)

)


end impvdm
```