

組込みコンポーネントシステム
TECS 仕様書

NPO 法人 TOPPERS プロジェクト
TECS ワーキンググループ

Draft 版 (V1.3.0.1)
2015 年 11 月 15 日

目次

0.1	序	3
0.1.1	著作権と利用条件	3
0.1.2	免責規定	4
0.1.3	ドラフト版について	4
0.1.4	注記について	4
0.2	改訂履歴	5
0.2.1	暫定第一版	5
0.2.2	ドラフト V0.1 (修正中)	5
第 1 章	TECS の概要	7
1.1	TECS 仕様の開発	7
1.1.1	名前の由来	7
1.1.2	TECS の開発母体	7
1.1.3	TOPPERS プロジェクトのリリースするカーネルとの 関係	8
1.1.4	目的	8
1.1.5	想定するユーザ	9
1.1.6	記述方法	10
1.1.7	対応する参照実装のバージョン	10
1.2	方針と開発物	11
1.2.1	方針	11
1.2.2	開発物	12
1.3	仕様の概観	13
1.3.1	組込みコンポーネントシステム TECS 仕様の概観	13
1.3.2	TECS による開発手順	14
1.3.3	TECS の設計領域	15
第 2 章	TECS コンポーネントモデル	17
2.1	基本モデル	17
2.1.1	セル (cell)	17
2.1.2	シグニチャ (signature)	17
2.1.3	セルタイプ (celltype)	18
2.1.4	シングルトン (singleton)	18

2.1.5	アクティブ (active)	18
2.1.6	受け口 (entry port)	18
2.1.7	呼び口 (call port)	18
2.1.8	結合 (join)	19
2.1.9	属性 (attr)	19
2.1.10	内部変数、または単に変数 (var)	19
2.1.11	ID と IDX	20
2.2	用語の補足説明	20
2.2.1	コンポーネントインスタンス	20
2.2.2	呼び元と呼び先	20
2.2.3	呼び口関数と受け口関数	21
2.2.4	合流	21
2.3	生成と結合に関するモデル	21
2.3.1	セルの生成	21
2.3.2	セル間の結合	21
2.3.3	オプション呼び口	22
2.3.4	メッセージ通信に関するモデル	22
2.4	関数とデータ型に関するモデル	22
2.4.1	データ型	22
2.4.2	データ型の取り込み	22
2.4.3	関数	23
2.4.4	引き数、戻り値	23
2.4.5	一方向関数 (oneway function)	23
2.4.6	データ型の逸脱 (deviate)	24
2.5	一対多の結合のモデル	24
2.5.1	呼び口配列	24
2.5.2	受け口配列	24
2.6	コールバックに関するモデル	24
2.6.1	コールバック	25
2.7	コンテキストに関するモデル	25
2.7.1	コンテキスト (context)	25
2.8	複合セルタイプのモデル	26
2.8.1	複合セルタイプ (composite celltype)	26
2.8.2	複合セルタイプにおけるシングルトンとアクティブ	26
2.9	静的 API に関するモデル	26
2.9.1	ファクトリ (factory)	26
2.10	リクワイアに関するモデル	27
2.10.1	リクワイア (require)	27
2.10.2	リクワイア呼び口	27

2.11	アロケータに関するモデル	27
2.11.1	アロケータ (allocator)	27
2.11.2	アロケータ呼び口	28
2.11.3	リレーアロケータ	28
2.11.4	デバイスアロケータ	28
2.12	プラグインに関するモデル	28
2.12.1	スループラグイン (through plugin)	29
2.12.2	接続 (connection)	29
2.13	リージョンに関するモデル	29
2.13.1	リージョン (region)	29
2.13.2	リージョン分割	29
2.13.3	複数のシングルトン	30
2.14	分散フレームワークのモデル	30
2.14.1	リモート呼び出し (Remote Procedure Call, RPC)	30
2.14.2	接続の分類	30
2.14.3	リモート呼び出しのモデル	31
2.15	検証支援機能に関するモデル	31
2.15.1	トレース	31
2.15.2	アサート	32
第 3 章	TECS コンポーネント図	33
3.1	TECS コンポーネント図の位置づけ	33
3.2	TECS コンポーネント図の記述方法	33
3.2.1	基本的なコンポーネント図	33
3.2.2	合流	33
3.2.3	呼び口配列	34
3.2.4	受け口配列	34
3.2.5	コールバック	34
3.2.6	呼び口配列と受け口配列	35
3.2.7	アクティブ	35
3.2.8	複合セル	36
3.2.9	RPC	36
3.2.10	アロケータ	37
3.2.11	多段リレーモデル	37
第 4 章	TECS コンポーネント記述言語 (TECS CDL)	41
4.1	定義	41
4.1.1	用語	41
4.1.2	文法定義	41
4.1.3	記述単位	42

4.1.4	文字コード	42
4.1.5	名前を扱うルール	42
4.1.6	前方参照	43
4.1.7	宣言と定義	43
4.1.8	名前・文字列の連結操作	44
4.2	字句構造	44
4.2.1	字句	44
4.2.2	空白文字	44
4.2.3	キーワード	45
4.2.4	指定子キーワード	45
4.2.5	識別子	45
4.2.6	リテラル	46
4.2.7	型名	47
4.2.8	コメント	47
4.3	名前有効範囲	48
4.3.1	一般名前有効範囲	48
4.3.2	シグニチャ関数名前有効範囲	49
4.3.3	仮引数名前有効範囲	49
4.3.4	セル属性名前有効範囲	49
4.3.5	タグ名前有効範囲	50
4.3.6	フィールド名前有効範囲	50
4.4	コンポーネント記述	50
4.4.1	コンポーネント記述	50
4.4.2	指定子文	50
4.4.3	文	51
4.4.4	文リスト	52
4.5	インポート (import) 文	52
4.5.1	インポート (import) 文	52
4.6	C 言語インポート (import_C) 文	53
4.6.1	C 言語インポート (import_C) 文	53
4.6.2	重複インクルード	54
4.7	文指定子	54
4.7.1	文指定子リスト	54
4.7.2	文指定子	54
4.7.3	コンテキスト指定子 (context)	55
4.7.4	逸脱指定子 (deviate)	56
4.7.5	アクティブ指定子 (active)	56
4.7.6	idx_is_id 指定子 (idx_is_id)	57
4.7.7	シングルトン指定子 (singleton)	57

4.7.8	アロケータ指定子 (allocator)	57
4.7.9	id 指定子 (id)	58
4.7.10	プロトタイプ指定子 (prototype)	58
4.7.11	プラグイン指定子 (generate)	58
4.8	型	59
4.8.1	型指定子修飾子リスト	59
4.8.2	型指定子	59
4.8.3	文字型	59
4.8.4	整数型	60
4.8.5	符号	60
4.8.6	符号付整数型	60
4.8.7	型修飾子	61
4.8.8	型名	61
4.9	構造体	61
4.9.1	構造体指示子	61
4.9.2	構造体宣言リスト	61
4.9.3	構造体タグ	62
4.9.4	構造体宣言	62
4.9.5	構造体宣言子リスト	62
4.9.6	構造体宣言子	62
4.10	ポインタ	62
4.10.1	ポインタ型	62
4.10.2	ポインタ指定子リスト	63
4.10.3	ポインタ指定子	63
4.11	列挙	63
4.11.1	列挙指定子	63
4.11.2	列挙型	63
4.11.3	列挙子リスト	63
4.11.4	列挙子	64
4.12	宣言子	64
4.12.1	宣言子	64
4.12.2	直接宣言子	64
4.12.3	宣言子リスト	65
4.13	パラメータ	65
4.13.1	パラメータタイプリスト	65
4.13.2	パラメータリスト	65
4.13.3	パラメータ宣言	66
4.13.4	パラメータ指定子リスト	66
4.13.5	パラメータ指定子	66

4.13.6	in 基本指定子	67
4.13.7	send 基本指定子	69
4.13.8	out 基本指定子	70
4.13.9	receive 基本指定子	72
4.13.10	size_is, count_is, string 指定子	73
4.13.11	size_is 指定子と count_is 指定子の特性	73
4.13.12	string 指定子の特性	75
4.13.13	typedef された型について	75
4.13.14	nullable 指定子	76
4.14	式	76
4.14.1	基本式	77
4.14.2	文字列リテラルリスト	77
4.14.3	後置式	78
4.14.4	単項式	78
4.14.5	単項演算子	78
4.14.6	キャスト式	78
4.14.7	乗除式	79
4.14.8	加減式	79
4.14.9	シフト式	79
4.14.10	関係式	79
4.14.11	等価式	79
4.14.12	and 式	80
4.14.13	exor 式	80
4.14.14	or 式	80
4.14.15	論理 AND 式	80
4.14.16	論理 OR 式	80
4.14.17	条件式	80
4.14.18	式	81
4.14.19	定数式	81
4.15	型定義	81
4.15.1	型定義文	81
4.15.2	型定義指定子	81
4.15.3	組込み型と typedef 型	82
4.15.4	組込み型	82
4.15.5	typedef 型	83
4.15.6	tecs.h と t_stddef.h	83
4.16	ネームスペース	83
4.16.1	ネームスペース文	84
4.16.2	ネームスペース名	84

4.16.3	ネームスペース識別子	84
4.17	シグニチャ	85
4.17.1	シグニチャ文	85
4.17.2	シグニチャ名	85
4.17.3	関数ヘッダリスト	85
4.17.4	関数ヘッダ	86
4.17.5	oneway 指定子	86
4.17.6	アロケータシグニチャ	87
4.18	セルタイプ	88
4.18.1	セルタイプ文	88
4.18.2	セルタイプ名	88
4.18.3	セルタイプ文リスト	88
4.18.4	指定子セルタイプ文	88
4.18.5	セルタイプ文指定子	89
4.18.6	アロケータリスト 2	89
4.18.7	セルタイプ文	90
4.19	呼び口、受け口	90
4.19.1	口文	90
4.19.2	口タイプ	91
4.19.3	ネームスペースシグニチャ名	91
4.19.4	口名	91
4.19.5	配列サイズ	92
4.19.6	固定結合	92
4.20	属性	93
4.20.1	属性文	93
4.20.2	属性宣言リスト	93
4.20.3	属性宣言	93
4.20.4	属性指定子	94
4.21	内部変数	94
4.21.1	内部変数文	95
4.21.2	内部変数宣言リスト	95
4.21.3	内部変数宣言	95
4.21.4	内部変数指定子	96
4.22	リクワイア	96
4.22.1	リクワイア文	97
4.23	ファクトリ	98
4.23.1	ファクトリ文	98
4.23.2	ファクトリ頭部	98
4.23.3	ファクトリ関数リスト	98

4.23.4	ファクトリ関数	99
4.23.5	ファクトリ関数名	99
4.23.6	ファクトリ引数リスト	99
4.23.7	名前置換	100
4.23.8	ファクトリヘッダ	100
4.24	セル	101
4.24.1	セル文	101
4.24.2	ネームスペースセルタイプ名	101
4.24.3	セル名	101
4.24.4	プロトタイプ宣言	101
4.24.5	純粋プロトタイプ宣言	102
4.24.6	指定プロトタイプ宣言	102
4.24.7	セルの id 番号	102
4.25	結合	103
4.25.1	結合リスト	104
4.25.2	指定子リスト付き結合文	104
4.25.3	結合指定子リスト	104
4.25.4	結合指定子	104
4.25.5	スルー指定子	104
4.25.6	プラグイン名	105
4.25.7	プラグイン引数	105
4.25.8	結合	106
4.25.9	呼び口の結合の場合	106
4.25.10	属性の初期化の場合（複合コンポーネントの内部セル の場合を除く）	107
4.25.11	複合コンポーネントの内部セルの属性の場合	107
4.25.12	逆結合	108
4.25.13	結合名	109
4.25.14	配列添数	109
4.26	複合セルタイプ	109
4.26.1	複合セルタイプ文	109
4.26.2	複合セルタイプ名	109
4.26.3	複合セルタイプ文リスト	110
4.26.4	指定子付き複合セルタイプ文	110
4.26.5	複合セルタイプ文指定子リスト	110
4.26.6	複合セルタイプ文指定子	110
4.26.7	複合セルタイプ文	111
4.26.8	複合セルタイプ口文	111
4.26.9	複合セルタイプ文指定子	111

4.27	複合セルタイプ属性	112
4.27.1	複合セルタイプ属性文	112
4.27.2	複合セルタイプ属性宣言リスト	112
4.28	内部セル	112
4.28.1	内部セル文	113
4.28.2	内部ネームスペースセルタイプ名	113
4.28.3	内部セル名	113
4.28.4	内部結合リスト	113
4.28.5	セル内外部結合文	114
4.29	外部結合	114
4.29.1	外部結合文	114
4.29.2	外部名	115
4.29.3	内部参照セル名	115
4.29.4	内部参照要素名	115
4.30	複合セルタイプ内のアロケータ	115
4.30.1	外部結合する場合	116
4.30.2	内部結合する場合	117
4.31	リージョン	117
4.31.1	リージョンの概要	117
4.31.2	リージョン文	118
4.31.3	リージョン指定子リスト	118
4.31.4	リージョン指定子	119
4.31.5	リージョンスルー指定子	119
4.31.6	リージョン属性	120
4.31.7	ノード (node)	120
4.31.8	リンク単位 (linkUnit)	121
4.31.9	ドメイン (domain)	121
4.31.10	クラス (class)	122
4.31.11	リージョン内部文	122
4.31.12	リージョン名	122
4.31.13	ネームスペースリージョン名	122
4.31.14	リージョンセル文	122
4.31.15	リージョンセル指定子リスト	123
4.31.16	リージョンセル指定子	123
4.31.17	指定リージョンのコード生成	123
4.31.18	複数のシングルトンセルタイプのセル	123
4.31.19	ルートネームスペースとルートリージョン	124
4.32	定数定義文	124
4.32.1	定数定義文	124

4.33	宣言	125
4.33.1	宣言文	125
4.33.2	宣言指定子	125
4.33.3	初期化宣言子リスト	125
4.33.4	初期化宣言子	125
4.34	初期化子	125
4.34.1	初期化子リスト	125
4.34.2	初期化子	126
4.34.3	C_EXP 初期化子	126
4.34.4	composite における C_EXP の名前置換	126
4.34.5	シグニチャプラグイン (generate) 文	127
第 5 章	TECS コンポーネント実装	129
5.1	定義	129
5.1.1	セルタイプコード	129
5.1.2	セルタイプヘッダ	129
5.1.3	グローバル名	129
5.1.4	インライン受け口関数	130
5.2	セルタイプコードに関する規則	130
5.2.1	使用言語	130
5.2.2	記述単位	130
5.2.3	ファイル名	130
5.2.4	インラインのセルタイプコードのファイル名	131
5.2.5	ファイルの記述内容	131
5.2.6	セルタイプヘッダのインクルード	131
5.2.7	参照できるもの	131
5.2.8	受け口関数の形式	132
5.2.9	受け口関数の形式 (受け口配列の場合)	133
5.2.10	CB ポインタ	133
5.2.11	呼び口関数	134
5.2.12	呼び口関数 (呼び口配列の場合)	135
5.2.13	属性	135
5.2.14	内部変数	136
5.2.15	非シングルトンセルタイプの場合のセルタイプコード の例	136
5.2.16	シングルトンセルタイプの場合のセルタイプコード	138
5.3	初期化コード	139
5.4	FOREACH_CELL マクロ	139
5.4.1	FOREACH_CELL マクロの使用	139
5.4.2	FOREACH_CELL マクロの多重使用	140

5.5	ファクトリ	141
5.5.1	コンフィギュレーションファイル	141
5.5.2	ファクトリヘッダ	142
付録 A	名前付け規則	143
A.1	接頭文字の規則	143
A.2	単語区切り	143
A.3	型情報	143
付録 B	マクロ	145
B.1	マクロ一覧	145
B.2	短縮形マクロ	145
B.3	通常形マクロ	147
付録 C	ファイルの一覧	149
C.1	ソースコードファイルの一覧	149
付録 D	TECS ジェネレータの出力	151
D.1	CB と INIB	151
D.1.1	CB と INIB	151
D.1.2	CB と INIB のコード	151
D.1.3	CB と INIB のバリエーション	152
D.2	初期化コード	152
D.2.1	初期化コードが必要となるケース	152
D.2.2	初期化コードの役割	153
D.2.3	初期化コードの生成と利用	153
D.2.4	初期化マクロ	153
D.2.5	初期化プログラム	154
D.2.6	全体初期化マクロ	154
D.3	データ構造のモデル	155
D.3.1	基本データ構造図	155
D.3.2	idx_is_id の場合のデータ構造図	155
D.3.3	受け口配列の場合のデータ構造図	155
D.3.4	呼び口配列の場合のデータ構造図	155
D.3.5	呼び口ディスクリプタ	156
D.3.6	受け口ディスクリプタ	156
D.3.7	受け口スケルトン関数	157
D.4	呼び口、受け口最適化	158
D.4.1	呼び口最適化の条件	158
D.4.2	呼び口最適化実施内容	159
D.4.3	結合状態と呼び口最適化実施内容	159

D.4.4	呼び口最適化（呼び口配列）	160
D.4.5	受け口最適化	160
D.4.6	受け口最適化実施内容	161
D.4.7	受け口最適化（受け口配列）	161
D.4.8	attribute 最適化	161
D.5	CB、INIB の最適化のモデル	161
D.5.1	CB、INIB の基本的なデータ配置	161
D.5.2	CB が存在しない場合	162
D.5.3	CB、INIB の最適化	162
D.6	Makefile	163
D.6.1	Makefile.tecsgen の内容	163
D.6.2	Makefile.depend の内容	164
D.6.3	Makefile.templ の内容	165
D.7	互換性	165
D.7.1	互換性	165
D.7.2	ソースコード互換性	166
D.7.3	バイナリコード互換性	166

組込みコンポーネントシステム TECS 仕様書

NPO 法人 TOPPERS プロジェクト

コンポーネント仕様ワーキンググループ編
暫定第一版 (V1.0.2.32)
copyright(c) 2008-2011 TOPPERS Project

0.1 序

0.1.1 著作権と利用条件

TECS Specification

Copyright (C) 2006-2015 by TOPPERS Project, Inc., JAPAN

上記著作権者は、以下の 1.~3. の条件を満たす場合に限り、本ドキュメント（本ドキュメントを改変したものを含む、以下同じ）を使用・複製・改変・再配布（以下、利用と呼ぶ）することを無償で許諾する。

1. 本ドキュメントを利用する場合には、上記の著作権表示、この利用条件および下記の無保証規定が、そのままの形でドキュメント中に含まれていること。
2. 本ドキュメントを改変する場合には、ドキュメントを改変した旨の記述を、改変後のドキュメント中に含めること。ただし、改変後のドキュメントが、TOPPERS プロジェクト指定の開発成果物である場合には、この限りではない。
3. 本ドキュメントの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者および TOPPERS プロジェクトを免責すること。また、本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも、上記著作権者および TOPPERS プロジェクトを免責すること。

0.1.2 免責規定

本ドキュメントは、無保証で提供されているものである。上記著作権者および TOPPERS プロジェクトは、本ドキュメントに関して、特定の使用目的に対する適合性も含めて、いかなる保証も行わない。また、本ドキュメントの利用により直接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

0.1.3 ドラフト版について

これまでの暫定版に対し以下を施しドラフト版とした。

- 暫定版は V1.0.2.24 までの仕様を規定したが、ドラフト版では V1.3.0.1 版を

組込みコンポーネントシステム TECS は、参照実装と仕様書を同時にリリースするが、仕様書は第一版として扱うには整備が不十分である。このため、本書は暫定第一版としてリリースする。後日改めて、第一版としてリリースしなおす予定である。

暫定一版では、記載すべき項目については、ほぼ網羅したつもりであるが、記載の細部については見直しが不十分である。

また、TECS の最も重要な機構の一つであるプラグインと、それにより実現される分散フレームワーク、トレース機能などについて、実装、仕様書ともにリリースに間に合わなかったため、暫定一版には含めなかった。いくらかの第一版に詰め込めなかった機能とともに、今後の改版にあわせて追記する予定である。

0.1.4 注記について

以下の隅付括弧により、注記を加える。

注記法	注記の意味
【参照実装における制限】	参照実装の TECS ジェネレータとの相違
【仕様決定の理由】	仕様を決定した背景などの説明
【使用上の注意】	仕様の誤解しやすい注意点などの説明
【補足説明】	仕様の理解を助けるための補足説明
【未決定事項】	仕様が未定の事項の説明
【制限】	仕様決定の際に考慮が不十分で制限の残る点の説明

0.2 改訂履歴

0.2.1 暫定第一版

2009年5月13日 新規作成

0.2.2 ドラフト V0.1 (修正中)

2015年9月13日 新規作成

暫定第一版から追加した内容は以下のとおりである。

- 前方参照 (セルの前方参照の仕様変更)
- シグニチャプラグイン文
- id 指定子 id
- プロトタイプ指定子 prototype
- プラグイン指定子 generate
- 組込み型と typedef 型
- tece.h と t_stddef.h
- 固定結合
- プロトタイプ宣言
- 純粋プロトタイプ宣言
- 指定プロトタイプ宣言
- セルの id 指定
- 逆結合
- リージョンスルー指定子
- リージョン属性 node, linkunit, domain, class
- ルートネームスペースとルートリージョン
- nullable 指定子

暫定第一版から変更した内容は以下のとおりである。

- リクワイア (シングルトンの解除)
- ネームスペース (タグ名有効範囲の変更)

- 文字型
- size_is, count_is 指定子の特性

第5章の見出しを「TECS コンポーネント実装モデル」から「TECS コンポーネント実装」に変更するとともに、TECS ジェネレータの生成コードに関する説明を付録に移した。

第6章 TECS RPC を削除した。

第1章 TECS の概要

1.1 TECS 仕様の開発

TECS は、組込みシステムに最適なソフトウェア部品化の仕組みを提供する。本章は、TECS の開発の動機、目標、方針を説明する。また、次章からの仕様の定義に先立って、TECS を概観する。

1.1.1 名前の由来

TECS は TOPPERS Embedded Component System の各単語の頭文字を取ったものである。

TECS を音を出して読む場合、「テクス」と発音する。

1.1.2 TECS の開発母体

TECS は NPO 法人 TOPPERS プロジェクトコンポーネント仕様ワーキンググループにより開発された。

本仕様の開発に関わったワーキンググループのメンバーを以下に記す。

安積 卓也 (名古屋大学)
石川 拓也 (名古屋大学)
鵜飼 敬幸 ((株) ヴィッツ)
大山 博司 (オークマ(株), ワーキンググループ主査)
河田 智明 (名古屋大学)
小南 靖雄 (フリーエンジニア)
高木 信尚 ((株) きじねこ)
高田 広章 (名古屋大学, NPO 法人 TOPPERS プロジェクト会長)
成瀬 有美 (名古屋大学)
原 拓 (名古屋大学)
山本 将也 (パナソニック アドバンステクノロジー(株))

(あいうえお順、所属は関与した当時のもの)

1.1.3 TOPPERS プロジェクトのリリースするカーネルとの関係

TECS は TOPPERS/ASP カーネルに最適化して設計されている。

しかし、その利用は TOPPERS/ASP カーネルに限定されるものではなく、TOPPERS プロジェクトの提供する統合仕様に準拠するカーネルでも、使用することができる。ただし、本書で定義する TECS の基本的な仕様についてであり、カーネルオブジェクトを TECS コンポーネントとして扱う場合には、それぞれのカーネルごとに異なる定義が必要である。

TOPPERS 統合仕様に準拠しない OS 上で動作させる場合には、型の扱いなどの基本部の相違が生じるが、TECS は C99 の仕様に適合するように仕様を策定しており、調整できないほどの困難が生じる可能性は、きわめて小さい。

1.1.4 目的

TECS の開発の動機であり目的は、以下に挙げる 3 つである。

1. 大規模、複雑化する組み込みソフトをコンポーネント構造にすることで見通しをよくする
2. コンポーネントモデルを規定することで、組み込み用ソフト部品の仕様定義を容易にし、流通を促進する
3. 組み込み向きの分散フレームワークなどフレームワークを実現するためのフレームワークを実現する

この 3 つを達成するには、広範な機能を持つシステムあるいはフレームワークが必要になる。TECS は、この 3 つを同時実現するための包括的な手段の提供を目的とする、コンポーネントシステムである。

上記の 1. について、今日、組み込みシステムの大規模化、複雑化が進展し、今後さらに進むことが予想される。組み込みシステムの多くは、継続的に製品改良が行われており、年々ソフトウェアに機能が加えられて大規模化、複雑化している。この結果、ソフトウェア構造は時とともにゆがめられ、理解しにくいものになっている。これを整理しなおすためのツールが必要であり、その提供が TECS の目的の一つである。

上記の 2. について、TOPPERS/ASP のような RTOS 向けのソフトウェア部品の種類が不足している問題が続いている。組み込みシステムに向けたソフトウェア部品としては TCP/IP が定義されているが、その他について標準化が進捗していない。今日各種の部品が必要になっているのに対し、標準化作業を行うだけの人的資源が不足している。標準化を提案するための基礎とし

て部品を構成するモデル、記述手段が必要になっており、その提供が TECS の目的の一つである。

上記の 3. について、これまで組込みシステム向けの分散フレームワークとしては、組込み CORBA のようなものが標準化提案されたことがあるが、これはホストコンピュータとの間での分散を実現することが主目的で、例えば RTOS のタスク間通信のような小さな領域での利用には不向きであった。また、マルチプロセッサ、マルチコアシステムが増加しているが、プロセッサ間、コア間のような比較的密接した領域に用いるオーバーヘッドの小さな分散フレームワークが期待される。

フレームワークとしての利用は、分散フレームワークにとどまらず、例えば mruby のような、より高水準の言語と連携するフレームワークとして用いることができる。この場合、TECS コンポーネントと mruby をブリッジするコンポーネントを自動生成させることで、TECS コンポーネントの従来と異なる活用が期待できる。

1.1.5 想定するユーザ

以下のようなユーザを想定する。

1. 大規模組込みシステムの開発者

システムをコンポーネント構造とすることで、システム全体の構造を見やすいものとするができる。規模の大きなシステムでは、多くの設計者が関わることになるが、各コンポーネントの役割を理解して、それぞれに設計し、結合してシステムを完成させることができる。

TECS はソフトウェアの検証を支援する機能を提供する。コンポーネント間の呼び出しを記録するログ機能による記録や、コンポーネントの呼び出しの事前条件、事後条件、不変条件を満たすかの検証ができる。これらの機能は、コンポーネントが期待した振舞いをしているかを検証するのに役立つ。複雑化、大規模化が進む一方で、高信頼化への期待も高まっているが、システムの見通しをよくすることは、同時に検証性を高めることにも役立つ。

【未決定事項】コンポーネントの呼び出しの事前条件、事後条件、不変条件を満たすかの検証機能は、予定されている仕様であり、現状の仕様には含まない。

1. ソフトウェア部品提供者

ソフトウェア部品を提供する場合、これまで標準的な部品の利用形態が定められていなかったが、部品の利用形態の共通化が図られて、部品の利用形態を説明するためのコストが低減できる。また TECS のコンポーネントは、既存のソースコードとの結合が容易である上にオーバーヘッドが極めて小さい。

さらに、上述したような検証機能が利用でき、部品が期待した動作をしない場合のサポートを容易化する。

このようにソフトウェア部品の提供者が、TECS 仕様に準拠したコンポーネントとして提供することで、多くのメリットが得られる一方で、実行速度、フットプリントともにオーバーヘッドが極めて小さいためにデメリットは小さい。

TECS コンポーネント記述言語によりソフトウェア部品の API を定義することで、実装と切り離すことができる。さらに C や C++ のそれよりも厳密にすることができる。将来的には、部品の API が標準化されて、ますます部品流通が促進されることが期待される。

1. マルチプロセッサシステム開発者

マルチプロセッサシステムの開発者にとっては、分散フレームワークが役に立つ。TECS の分散フレームワークは、コンポーネント間に通信チャンネルコンポーネントを挿入することで実現される。

通信チャンネルコンポーネントは、マーシャラ、アンマーシャラ、チャンネルなどのコンポーネントにさらに分解される。このチャンネルを交換可能とすることで、種々の環境に適用させることが容易である。

分散フレームワークでは、呼び出しオーバーヘッドが大きいいため、一般に結合が疎になるように設計する。一方、コンポーネントすなわちソフトウェアモジュール間の結合を疎にすることで、組合せを変更した再利用が容易になる。この両者の方向性が一致していることから、分散フレームワークとコンポーネントシステムを同一のシステムで実現することが容易である。

1.1.6 記述方法

本書では、しばしば後から定義するものを参照するが、一々後述部分を参照とは記載しない。

【補足説明】前から順に読み進めて未定義の語が出現した場合、後述部分に記載がある。

1.1.7 対応する参照実装のバージョン

本書の記載は、TOPPERS プロジェクト TECS WG にて開発された TECS ジェネレータ V1.3.0.0 に対応する。ただし、本書の記載と相違する点がある。そのような点については、注記を入れた。

1.2 方針と開発物

1.2.1 方針

TECS の仕様を設計するにあたって、組込みシステムに適したコンポーネントシステムを実現するために、またコンポーネントシステムによって解決可能な課題をいくつも検討し、設計の方針とした。以下に、設計の方針を記す。

1) コンポーネントモデル

- 静的なコンポーネント生成と結合する（オーバーヘッドの最小化、必要なものは予め全て用意しておく）
- 結合は関数結合のみとする
- すべてのものをコンポーネントとする（アロケータ、RPC チャネルなどを隠さない）
- 再利用以外の開発効率改善機能の充実を図る（具体的には RPC やテスト支援機能）

2) コンポーネント記述言語

- コンポーネント記述言語は、TECS 仕様として定義する
- TECS 仕様のコンポーネント記述言語を TECS CDL と呼ぶ
- コンポーネント記述言語により、主にはインタフェース定義、コンポーネント定義、組上げを記述する
- コンポーネント記述言語では、コンポーネントの実装は記述しない

3) 実装言語

- 実装言語として C 言語を基本とする（想定する利用者の知識）
- インタフェース定義の曖昧さをなくす（型、入出力など）
- オブジェクト指向技術を直接的には取り入れない
- C 言語のマクロ定義により呼び先を変更する手段を実現する
- コンポーネントはソースコード供給を基本とする
- （ネームスペースによる名前衝突の回避）

4) ジェネレータと組込み支援

- 組上げ
- ジェネレータによるコンポーネント間のインタフェースコードの生成

- 柔軟な組上げと最適化によるオーバーヘッドの最小化
 - ROM/RAM 支援
 - コンテキスト
- 5) フレームワーク実現インタフェース
- プラグインによるフレームワーク・フレームワークの実現
 - 分散フレームワークの実現
 - (分散と非分散で同じコード)
 - 多段リレーモデル
 - テスト支援機能の実現 (トレース機能)
 - 排他制御コードの自動生成
- 6) 型
- C 言語定義された型を取込むことができる
 - TOPPERS/ASP で標準使用する型を TECS の標準的な型とする
 - ビット長を明確化する . ただし、上記のために明確でないものも標準利用する
- 7) データ受け渡し
- 関数呼び出し時のデータの受渡しのメモリ領域は呼び元で準備する
 - もしくは、メモリ領域をデータを渡す側がアロケータで確保し、渡された側が解放する

1.2.2 開発物

組込みコンポーネントシステム TECS 仕様として、ソフトウェアコンポーネントの構成方法、使用方法を規定する。より具体的には、以下のものが定義される。

- TECS コンポーネントモデル
- TECS コンポーネント図
- TECS コンポーネント記述言語 (TECS CDL)
- TECS コンポーネント実装

TECS コンポーネントモデルは、TECS によるソフトウェア部品の構成の仕方と使い方を規定するものである。

TECS コンポーネント図は、TECS コンポーネントモデルを図により表現する手段を与えるものである。

TECS コンポーネント記述言語は、正確なコンポーネントの定義とコンポーネントを組合わせてシステムを組上げる記述をするものである。

TECS コンポーネント図は、TECS コンポーネント記述言語に比べ組上げが中心であり、コンポーネントを正確に表現する手段を持たず、補助的な利用になる。

TECS コンポーネント実装は、振る舞いのコード (C 言語で記述する) の書き方を規定する。

本仕様書では、第 2 章で TECS コンポーネントモデル、第 3 章で TECS コンポーネント図を、第 4 章で TECS コンポーネント記述言語を、第 5 章で TECS コンポーネント実装を定義する。

【補足説明】本書の暫定版では TECS コンポーネント実装を「TECS コンポーネント実装モデル」と呼んでいたが、ドラフト版からは TECS コンポーネント実装と改めるとともに、TECS ジェネレータの生成するコードは、TECS 仕様の本文から外し、付録に移した。

1.3 仕様の概観

1.3.1 組込みコンポーネントシステム TECS 仕様の概観

次章からの TECS 仕様の定義に入る前に TECS コンポーネントの仕様を概観する。

TECS コンポーネントモデルでは、ソフトウェア部品すなわちコンポーネントのことをセルと呼ぶ。

セルは関数インタフェースを持つことができ、関数インタフェースを介して他のコンポーネントと結合し、組上げて用いられる。

関数インタフェースには、機能を提供するための受け口と、そのコンポーネントが機能するために必要となる外部の機能を利用するための呼び口がある。

関数インタフェースは、関数ヘッダの組により識別される。この関数ヘッダの組のことをシグニチャと呼ぶ。

同じシグニチャを持つ呼び口と受け口を結合することができる。

セルは、属性と内部変数を持つ。TECS のコンポーネントは ROM 化システムで利用されることが考慮されていて、属性は ROM に置かれ、変数は RAM に置かれることが仮定される (すべてを RAM へロードして動作する場合には、属性も RAM に置かれる)。

セルは、セルタイプに属す。セルタイプはソフトウェア部品の型式を定義するものである。同じ型式のソフトウェア部品は、同じ関数インタフェースを備え、同じ属性、変数を持ち、同じ機能を果たす。

セルのことをコンポーネントインスタンスと呼ぶことがある。これは、セルタイプのことをコンポーネントと捉えた場合に、セルのことを表す。

いくらかの組込みコンポーネントシステムにみられる、時間制約やフットプリントの制約を扱うための手段は、少なくとも現在のバージョンの TECS の範囲外である。

TECS では、以下のものを TECS コンポーネント記述言語 (TECS CDL) により記述する。

- コンポーネントの定義、すなわちシグニチャやセルタイプの定義
- コンポーネントを組合わせたシステム構築 (組上げと呼ぶ) すなわちセルの配置とセル間の結合

上記にない要素、すなわちセルの振舞いについては、セルタイプごとに、C 言語により記述する。

1.3.2 TECS による開発手順

図 1.1 に TECS による開発手順を示す。この手順は、標準的な開発手順であり、既存のコードにコンポーネントを結合する場合など、適宜変更して開発を行う。

始めに、(1)TECS コンポーネント図でソフトウェア構造を表現する。

次に必要となるソフトウェア部品について、TECS CDL を用いて、部品の型である (3) セルタイプを記述する。セルタイプを定義する前に (2) シグニチャを記述しておく必要がある。

必要な部品の型が揃ったら (4) 組上げ記述を行う。組上げとは、セルを並べてセル間の結合を行うことである。

シグニチャ記述、セルタイプ記述、組上げ記述ができましたら、これらを (6)TECS ジェネレータに通す。そうすると (10) インタフェースコード、(9) ヘッドファイル、(8) セルタイプコードのテンプレートが生成される。

セルタイプコードのテンプレートを元に、セルタイプコードを記述する。セルタイプコードとは、セルの振舞いを規定するもので、C 言語で記述する。

インタフェースコードとセルタイプコードを、それぞれコンパイルし、リンクすることでアプリケーションモジュールが完成する。これをターゲットデバイスに組み込んで製品ができあがる。

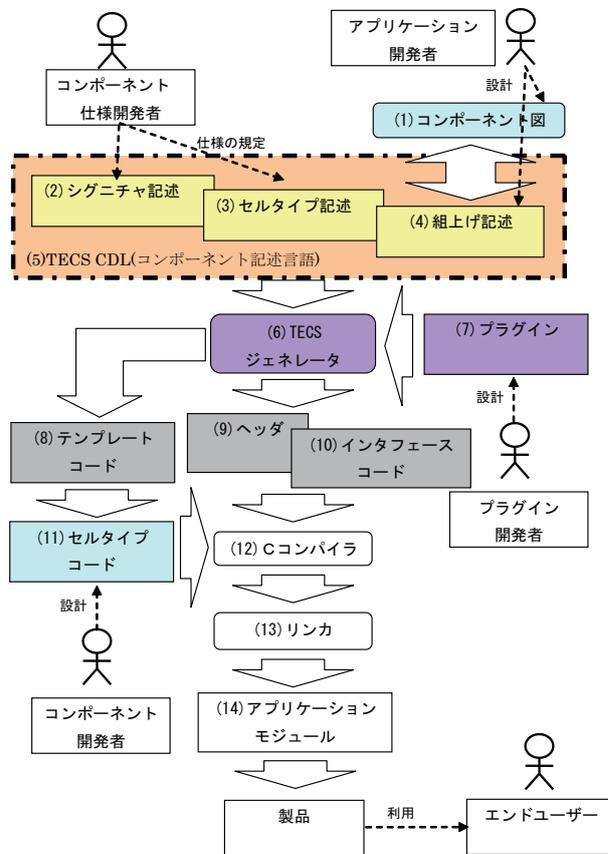


図 1.1: TECS による開発手順

1.3.3 TECS の設計領域

TECS は開発の中流段階で用いられることが想定される。

図 1.2 は、TECS の設計領域を図に表したものである。

TECS は、分析・設計の終わりの段階から、実装、検証の工程に用いることができる。UML などの手段を分析・設計を用いて行った後、TECS コンポーネント図としてソフトウェア構造を表し、TECS コンポーネント記述言語 TECS CDL により正確なコンポーネント記述を行うとともに、TECS ジェネレータで生成されたテンプレートコードを元に C 言語でコンポーネントの振舞いの実装を行う。

C 言語やアセンブラが実装の全体をカバーできるのに対し、TECS はソフトウェア構造を表すものであり、構成するコンポーネントとそれらを結合する部分に限定される。

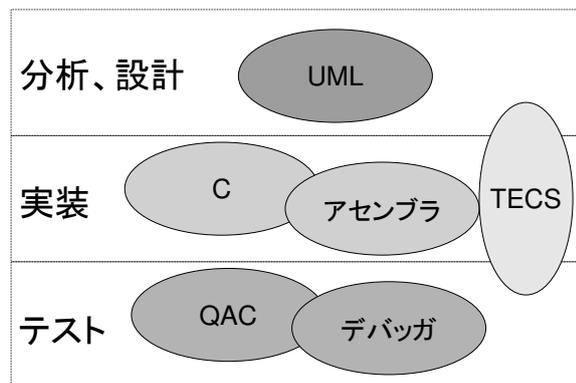


図 1.2: TECS の設計領域

第2章 TECS コンポーネントモデル

本章では、TECS コンポーネントモデルを定義する。
TECS コンポーネントモデルは、以下の定義からなる。

- TECS 仕様に基いて作製されるコンポーネントの持ちうる要素、すなわち構成要素など、TECS コンポーネントの持つ特性
- TECS コンポーネントを組合わせてシステムを組上げる方法

2.1 基本モデル

基本モデルとして、TECS コンポーネントの基本的な構成要素を定義する。

2.1.1 セル (cell)

ソフトウェア部品、すなわちコンポーネントインスタンスのことをセルと呼ぶ。セルは、以下のものを持つことができる。

- 受け口
- 呼び口
- 属性
- 内部変数

セルは、いずれかのセルタイプまたは複合セルタイプに属する。
セルは、セル名を持つ。

2.1.2 シグニチャ (signature)

シグニチャは、関数ヘッダの集合によりセル間のインタフェースを規定するものである。呼び口や受け口はシグニチャに対応付けられる。

シグニチャは、シグニチャ名を持つ。

2.1.3 セルタイプ (celltype)

ソフトウェア部品の型式をセルタイプと呼ぶ。セルの持つ各要素（呼び口、受け口、属性、内部変数）のシグニチャや型は、セルタイプにおいて定義される。また、セルの動作を実装するコードは、セルタイプごとに記述される。セルの動作を実装するコードを、セルタイプコードと呼ぶ。

セルタイプは、セルタイプ名を持つ。

【補足説明】セルタイプは、ソフトウェア部品の型式を定義するものであり、セルはその型式のソフトウェア部品にあたる。

2.1.4 シングルトン (singleton)

セルタイプに属するセルを一つだけに制限する、シングルトンとすることができる。シングルトンのセルタイプおよびセルのことを、それぞれシングルトンセルタイプ、シングルトンセルと呼ぶ。

2.1.5 アクティブ (active)

アクティブは、能動的に動作するセル、すなわちタスク、ハンドラなど CPU 資源の割付けを伴うセルそのものであるか、それらを少なくとも一つ内包したセルのことである。

セルがアクティブであるかどうかは、セルタイプに指定され決定される。

アクティブセルは、受け口を一つも持たなくてもよい。あるいは、アクティブセルの持つすべての受け口が未結合であってもよい。逆にアクティブセルでなく、受け口を持たない、あるいは、セルの持つすべての受け口が未結合であって、かつファクトリも持たない場合、そのセルは呼び出されて動作することはなく、誤りである。

2.1.6 受け口 (entry port)

受け口は、セルが機能を提供するための口である。受け口はシグニチャに対応付けられており、シグニチャの関数群によって機能を提供する。

受け口は、受け口名を持つ。

受け口の関数は、インライン実装することができる。

2.1.7 呼び口 (call port)

呼び口は、セルが他のセル（自セルも可）の提供する機能を利用するための口である。呼び口はシグニチャに対応付けられる。

呼び口は、呼び口名を持つ。

呼び口は、必ずいずれかの受け口と結合されなくてはならない。しかし、オプションと指定することで、受け口に結合しないままとすることができる。

2.1.8 結合 (join)

セルの呼び口と受け口を結びつけることを結合と呼ぶ。呼び口と受け口を結合する場合、それぞれ同じシグニチャに対応付けられていなくてはならない。

結合は、呼び元のセルにおいて指定する。

【仕様決定の理由】ハードウェア部品の結合においては、結合が部品の端子間を結ぶ線材に対応付けられるために、結合を示す情報をセル外で指定するのが妥当である。しかし、ソフトウェア部品の結合では、呼び元から呼び先へのポインタ情報を、呼び側のセルに埋め込むことにより実現される。このため結合を呼び側セルにおいて指定することが妥当である。

【補足説明】シグニチャの一致は、形式的に一致していて、2つのセルがつながることを保証する。しかし、そこに流れる信号（関数の仕様）が一致するかどうかについては、保証しない。シグニチャを定義する際に、関数がどのような機能を果たすのかを明確にし、結合可能性を明確にしなくてはならない。

2.1.9 属性 (attr)

属性は、セル内部に値を保持するものである。属性の持つ値は、セルの生成後に変更することのできない定数である。このため属性は、実装言語において、定数として扱われる。

実装言語から参照しない属性の指定子として `omit` を指定できる。そのような属性はファクトリで用いられることが想定される。

【補足説明】属性は ROM 化システムでは ROM に置かれることが想定される。汎用 OS のように RAM にダウンロードして動作するように実装する場合には、属性は RAM に置かれることになるが、定数であることには変わりがない。

2.1.10 内部変数、または単に変数 (var)

内部変数も、セル内部に値を保持するものである。内部変数の値は、動的に変更することができる。

内部変数の初期値は、個別に指定することができない。しかし、初期値に属性を含めることができ、間接的には個別に異なる初期値を与えることができる。

【仕様決定の理由】内部変数は RAM に置かれることが想定され、ROM 化システムにおいては初期値を個別に与えると、初期値を保持するために ROM 領域を多く消費してしまう。これを避けるために個別には指定できないようにした。

【補足説明】内部変数は、混乱を生じない場合には単に変数と呼ぶことがある。

size_is 指定されたポインタ型の内部変数は、size_is の引数で指定された大きさの配列変数の指定とみなされる。この場合、ポインタ型の変数は属性の一つとして扱われ、配列変数が内部変数の一部として扱われる。

2.1.11 ID と IDX

セルには、整数の ID が割付けられる。IDX はセルを指すポインタ情報または ID である。通常 IDX として、セルを指すポインタ情報が用いられる。

【未決定事項】セルの IDX を整数の ID とした場合の使い方について、有効な利用方法が想定されていない。

2.2 用語の補足説明

前節でコンポーネントモデルを説明する際に使用した用語で未定義かつ分りにくいもの、今後使用する用語について、以下に説明を加える。

2.2.1 コンポーネントインスタンス

具体的なソフトウェア部品をコンポーネントインスタンスと呼ぶ。単にコンポーネントと呼んだ場合、ソフトウェア部品の型式、すなわちセルタイプを指すか、具体的な部品、すなわちセルのことを指すか曖昧になる。具体的な部品すなわちセルのことを指し示す場合には、コンポーネントインスタンスと呼ぶ。

2.2.2 呼び元と呼び先

結合されたセルにおいて、呼び口の側のセルを呼び元、受け口の側のセルを呼び先という。

【補足説明】TECS コンポーネント間の結合は、すべて関数結合である。呼び元、呼び先は、それぞれ関数の呼び元、関数の呼び先である。

2.2.3 呼び口関数と受け口関数

受け口関数とは、受け口に対応付けられたシグニチャの持つ関数について、セルタイプの受け口の実装として記述された関数のことである。

あるセルタイプのすべての受け口関数を記述したものが、セルタイプコードである。

呼び口関数とは、呼び口に対応付けられたシグニチャの持つ関数で、セルタイプコードの中から呼び出し可能な関数のことである。

2.2.4 合流

結合において、複数の呼び口からの線を合流させて一つの受け口につなぐことができる。この逆に、一つの呼び口を複数の受け口に対応させる、分流はできない。

【補足説明】ハードウェアコンポーネントとの対比では、呼び口が関数を呼び出す側であり出力に対応付けて考えられる。また受け口は関数が呼び出される側（呼び出しを受け入れる側）であり入力に対応付けて考えられる。しかし、複数の呼び口をまとめて一つの受け口に結合できるが、反対に一つの呼び口から複数の受け口に分流させて結合することはできない。

2.3 生成と結合に関するモデル

TECS は静的な生成と結合を基本とする。また、TECS のセル間の結合に関するモデルは、関数結合のみである。

2.3.1 セルの生成

セルは、静的に生成する。静的に生成するとは、コンパイル時に生成されて実行モジュールにセルのイメージが格納されることである。

【制限】現在の仕様では動的な生成について、未検討である。将来の拡張とする。

2.3.2 セル間の結合

結合は静的に行う。静的に結合するとは、コンパイル時に結合されて、実行モジュールのセルのデータとして格納されることである。結合には、後で説明するスループラグインを指定することができる。

なお、結合にセルの属性に値を割付けることを含める場合がある。

2.3.3 オプション呼び口

オプションと指定された呼び口は、受け口に結合しないままとすることができる。通常、呼び口はいずれかの受け口に結合しなくてはならないが、オプションと指定することにより、この制限を解除することができる。

このような呼び口は、結合されていない可能性があるため、実行時に結合されているかどうかチェックする必要があるが生じる。

2.3.4 メッセージ通信に関するモデル

TECS 仕様としては、直接的にメッセージ通信に関する機構を提供しない。

メッセージ通信は、メッセージ通信チャンネルコンポーネントをコンポーネント間に置くことにより実現する。

【仕様決定の理由】メッセージ通信による結合を実現するには、メッセージ通信機構が必要になるが、TECS ではメッセージ通信機構もコンポーネントとして扱い、メッセージ通信機構との間は関数結合する。こうすることで、アプリケーションに適したメッセージ通信機構を設けることができる。また TECS ではオーバーヘッドが小さいためメッセージ通信機構をコンポーネントとして扱うことのデメリットはない。

2.4 関数とデータ型に関するモデル

2.4.1 データ型

TECS のデータ型は、C 言語のデータ型に直接的に対応付けられる。

ただし、基本的には `int8_t`, `int16_t`, `float32_t`, `double64_t` のようにサイズが明確になるものの使用が推奨される。

`int`, `short`, `long` のようにサイズが曖昧になるものの使用は推奨しないが、C 言語実装との親和性に配慮して使用可能とする。

【仕様決定の理由】TECS のデータ型は、TOPPERS/ASP あるいは、その派生版のカーネルとの親和性を最大限に考慮した仕様とした。また TECS とのデータ型を合わせるために TOPPERS/ASP のデータ型の見直しも行われた。これは TOPPERS/ASP の一般公開の初版から対応済みである。

C 言語のポインタ型は、配列を指す場合と非配列を指す場合とがある。TECS では、配列の場合 `size_is`, `count_is`, `string` を指定して明示する必要がある。

2.4.2 データ型の取り込み

TECS のデータ型は、C 言語のデータ型に直接的に対応付けられる。このため C 言語のヘッダファイルを解釈して、データ型を取り込みむことを可能

とする。

2.4.3 関数

シグニチャに記述する関数ヘッダは、C 言語における関数として本体が実装されることを想定する。しかし、C 言語の関数インタフェースでは、引き数の型や入出力方向の曖昧さが、しばしば問題となる。TECS では、曖昧さのない関数インタフェースを用いる。

2.4.4 引き数、戻り値

TECS の関数では、ポインタ型の引き数の入出力方向、配列か非配列か、アロケータにより確保されたメモリ領域であるかどうかを明示的に示す。

引き数には in, out, inout, send, receive のいずれかを指定する。in と send は呼び先の関数への入力を表す。out と receive は呼び先の関数からの出力を表す。inout は呼び先の関数への入出力を表す。

in と send、out と receive の違いは、ポインタ引き数を渡すとき、ポインタの指すメモリ領域の相違である。send と receive を指定された引き数は、アロケータにより確保されたメモリ領域を渡して、受け取った側で解放する必要があることを示す。

引き数の曖昧さがなくにより、関数呼び出しをパッケージ化するマージャラや、関数呼び出しをトレースするセルを自動生成することが可能となる。

関数の標準的な戻り値としては ER 型または ER_INT 型を推奨する。ER 型は、TOPPERS 統合仕様により規定される。

【仕様決定の理由】ER 型や ER_INT 型は TOPPERS/ASP や ITRON 仕様強く依存するものになる。他のシステムへの移植性は低下するが、分散フレームワークやアロケータではエラー値を返す必要なためである。しかし、この決定は、他のシステムへ TECS を適用する際に、それほど大きな障害となるほどのものではない。

2.4.5 一方向関数 (oneway function)

関数が戻り値を返さない、出力引数を持たない場合、一方向関数として扱うことができる。一方向関数は、リモート呼び出しにおいては非同期呼び出しとすることができる。

非同期呼び出しとは、呼び出された関数の処理が完了する前に、呼び出した関数が処理を再開することをいう。リモート呼び出しでない場合、一方向関数を含めて、関数呼び出しは同期呼び出しであり、呼び出された関数の処理が終わるまで待ってから、呼び出した関数の処理が再開される。

2.4.6 データ型の逸脱 (deviate)

TECS はデータ構造が客観的に判断できない、C 言語の曖昧なデータ型の使用を推奨しない。このため、関数の引き数のポインタの多重度は制限される。この多重度は、引き数の入出力方向ごとに規定される。これを満たさない場合、逸脱となる。シグニチャにおいて逸脱を宣言し、逸脱を明示する。逸脱したシグニチャは、特別に対応している場合を除きプラグインで挿入するセルを生成させることができない。

2.5 一对多の結合のモデル

TECS の結合では、呼び口から受け口への合流はできるが、呼び口の分流はできない。

分流では、同一の情報を一度に伝達したい場合と、選択したどれか一つのみ伝達したい場合とがある。同一の情報を伝達したい場合であっても、呼び出し順序が問題になる可能性がある。

呼び口配列は、分流を代替する手段である。

2.5.1 呼び口配列

呼び口配列は、配列化された呼び口である。配列添数により呼び先のセルを切替える。

呼び口配列によって、呼び口を分流させることができる。この場合、呼び元のセルでは、使用する口、すなわち添数を指定して呼び出す。

2.5.2 受け口配列

受け口配列は、配列化された受け口である。配列添数により呼び元セルを識別する。

受け口配列は呼び口配列と対に用いることで、複数の呼び元に対するコールバックに対応できる。

2.6 コールバックに関するモデル

TECS 仕様では明示的なコールバック機構を持たない。

【補足説明】ITRON 仕様におけるコールバックのように、実行時に関数ポインタを受け渡すことで結合先が決定されるものは、TECS では動的な結合に位置づけられる。

2.6.1 コールバック

コールバックは、2つのセルが、それぞれ受け口と呼び口を持ち、相互に接続し合っている状態、すなわちコールバックのある結合の例において、補助的な結合をコールバックと呼ぶ。なお、主体的な結合の名称は定められていない。この場合、主体的な結合において、呼び元と呼び先を区別し、コールバックでは呼び先から呼び元への呼び出しが行われる。

TECS の仕様としては、コールバックをサポートする機構を持たない。コールバックを主体的な結合とは別の、明示的な結合を記述する。

【仕様決定の理由】1. 呼び口と受け口を結合するだけで、コールバックの結合が生じるようにすると、合流してくる場合に、どこにコールバックすればよいかの決定ができない

【仕様決定の理由】2. コールバックがしばしば主体的な結合とは異なるコンテキストで呼び出される。例えば、デバイス进行操作するセルの場合、コールバックは割込みルーチンから呼び出されることになる。コンテキストの相違を明示する立場からは、このようなコールバックは不適切なものになる。

【仕様決定の理由】3. コールバックの結合は ITRON のような OS においても動的に行うが、少なくとも現バージョンの TECS としては、静的な結合のみが可能である。

【仕様決定の理由】4. 当初の目標として分散する場合と、しない場合で、コンポーネントが同じように動作することが期待された。コールバックの呼び出しを処理する手段が何通りか考えられるが、当初目標を達成するには何通りかの実装を用意しなくてはならない。

受け口配列と呼び口配列を対にして、複数のセルに対するコールバックを実現することができる。(次章のコンポーネント図を参照)

2.7 コンテキストに関するモデル

TECS では、ITRON のような OS のコンテキストを扱うため、コンテキストに関するモデルを持つ。

【制限】TECS でのコンテキストの扱いは、備忘としての扱いにとどめられている。

2.7.1 コンテキスト (context)

ITRON 系の OS において、コンテキストとしては、タスク部、非タスク部がある。コンテキストによって呼び出すことのできる API が異なるため、区別して扱う必要がある。TECS では、シグニチャ単位でコンテキストを指定する。

2.8 複合セルタイプのモデル

複合セルタイプは、コンポーネントを組み合わせた、新しいセルタイプを生成するものである。

2.8.1 複合セルタイプ (composite celltype)

複合セルタイプは、複数のセルを組合わせて、複合機能を備えた新しいセルタイプを生成するものである。複合セルタイプの内部のセルを内部セルと呼ぶ。複合セルタイプから生成されるセル、すなわち複合セルタイプに属するセルを複合セルと呼ぶ。

複合セルタイプは、属性、呼び口、受け口を持つ。これらは、複合セルを構成する内部セルに結合する。複合セルタイプに属するセルは、複合セルタイプを構成するセルのコピーとして展開される。

2.8.2 複合セルタイプにおけるシングルトンとアクティブ

複合セルタイプを構成するセルのどれか一つがシングルトンセルタイプに属する場合、複合セルタイプはシングルトンセルタイプとなる。

複合セルタイプを構成するセルのどれか一つがアクティブセルタイプに属する場合、複合セルタイプはアクティブセルタイプとなる。逆にアクティブな複合セルタイプの内部に、一つもアクティブな内部セルがない場合、誤りである。

2.9 静的 API に関するモデル

TECS は TOPPERS 統合仕様あるいは ITRON 仕様の静的 API を扱うための機能が備わっている。ファクトリは、セルタイプやセルが生成されるごとに、セルタイプの設計者が意図した何らかの文を、ファイルに出力するものである。ファクトリ機能は、静的 API を扱うことに限定されない。

2.9.1 ファクトリ (factory)

ファクトリは、セルタイプまたはセルの生成に伴って、ジェネレータが生成する標準的なコードとは別に、セルタイプで指定されたコードを生成する。具体的には、ITRON 仕様におけるコンフィギュレーションファイルへの静的 API の呼び出し文の生成に適する。

2.10 リクワイアに関するモデル

あるセルタイプのセルが、必ず結合されるセルへの結合をリクワイアとして指定できる。

2.10.1 リクワイア (require)

リクワイアは、共通して使われるものについて、結合を明示することなく使用できるものである。

例えば OS の API が、この使用に適する。

【仕様決定の理由】リクワイアは、OS の API やライブラリなどをセルタイプが要求することを明示する目的で導入が検討されたが、メモ的である（明示したことの効果を TECS の段階で発揮できない）。このため、現状では上記のような案となっている。

2.10.2 リクワイア呼び口

リクワイアによって指定される結合によって、呼び口が生成される。これをリクワイア呼び口という。

2.11 アロケータに関するモデル

TECS では組み込みシステムの特性を考慮して、アロケータを使い分けることが考慮されている。

【補足説明】汎用 OS 上のコンポーネントシステムであればアロケータは、OS やライブラリの提供するもので、コンポーネントシステムにおいては、既に存在するそれらを利用することがになる。

2.11.1 アロケータ (allocator)

アロケータは、メモリの動的な割付を行うものであるが、TECS においては引数を渡す場合のアロケータの使用について規定する。セルタイプコード内に閉じたアロケータの使用については TECS の仕様としては規定しない。

関数の呼び元から呼び先にアロケータから獲得したメモリ領域を渡す場合（入力引数）、反対に呼び先から呼び元にアロケータから獲得したメモリ領域を渡す場合（出力引数）、データを渡そうとする側がアロケータによりメモリ領域を確保し、渡された側が解放しなくてはならない。

このため、アロケータの使用にあたっては、データを渡そうとする側と渡された側で、どのアロケータを使用するか合意が必要となる。

シグニチャを定義する段階では、send/receive 引数において、使用するアロケータのシグニチャを定義する。

受け口側のセルを定義する段階で、具体的にどのアロケータを使用するかを定義する。

受け口側に定義する理由は、複数の呼び口が合流して受け口に結合される場合、すべてにおいて同じアロケータが使用されなくてはならないためである。

【補足説明】呼び側の関数がメモリ領域を確保して、やはり呼び側の関数がメモリを解放する場合には、アロケータに関する合意が不要であるので、アロケータの利用を TECS の範囲では扱わない。

2.11.2 アロケータ呼び口

呼び口または受け口のシグニチャによってアロケータが指定されている場合、アロケータセルへの結合が生じる。

アロケータセルへは、アロケータ呼び口を介して結合される。アロケータ呼び口は内部的に暗黙的に生成される。

2.11.3 リレーアロケータ

リレーアロケータは、send 指定された引き数としてセルの受け口に渡されたメモリ領域を、呼び口から呼び先のセルに引き渡す場合に用いるものである。

2.11.4 デバイスアロケータ

デバイスアロケータは、デバイスによって、あるいはそのドライバによってメモリ割付けできる領域が限定される場合に用いるものである。

この場合、デバイスのセルタイプにおいてアロケータの受け口も持ち、その受け口を通してアロケータが利用される。

【未決定事項】デバイスアロケータについては、コンポーネント記述、実装方法ともに未検討である。

2.12 プラグインに関するモデル

TECS では、分散フレームワークの実装など、セル間の結合に挿入する方式で実現する。挿入するセルは、プラグインによって自動生成させることができる。

【補足説明】現時点では、プラグインは、セル間の結合に挿入するスループラグインのみである。

2.12.1 スループラグイン (through plugin)

スループラグインは、その指定により、シグニチャ、セルタイプ、セルなどを生成する。

スループラグインを指定するには、セルの呼び口を結合する際に指定する `through` 指定子による方法、リージョンに指定する `in_through`, `to_through`, `out_through` 指定子による方法がある。これらはいずれも、接続された呼び口と受け口の間にセルを挿入する。

挿入されるセルのことをスルーセルと呼ぶ。

2.12.2 接続 (connection)

呼び口を受け口とつなぐことを結合と呼ぶが、呼び口と受け口の間にスループラグインによるセルを挿入する場合、通常の結合とは区別して接続と呼ぶ。

挿入されるセルには RPC プラグインによる RPC チャンネルなどがある。

2.13 リージョンに関するモデル

2.13.1 リージョン (region)

リージョンは、セルの配置を指定するものであり、セルのみが属することができる。リージョンは、以下の実現を目的とする。

- 1) リージョン間の結合を制限する
- 2) リージョン間の結合においてセキュリティ保護のためのセルを挿入する
- 3) 別リンクによる実装のためにリージョン毎にコードを生成する
- 4) 分散 (別プロセッサ) のためのリージョンに分け、結合においてはリモート呼び出しチャンネルセルを挿入する
- 5) 異なるリージョンにシングルトンセルタイプのセルを生成する (例えば分散でカーネルが複数存在する場合)
- 6) ローダブルモジュールをリージョンとして扱う

【参照実装における制限】現状では 6) を実現するためには、C コンパイラ後のリンクを解決する方法、`to_through` による生成コードの `region` 分けなどが必要。

2.13.2 リージョン分割

リージョンごとに分割してアプリケーションモジュールを構築することができる。この場合、リージョン間の結合、すなわちリージョンを超えたセルの結合はできない。

ただし、リージョン間の接続はできる。リージョン間にスループラグインによって生成されるコンポーネントが挿入されるとき、挿入されたコンポーネントがリージョン分割に対応していて、呼び元リージョンと呼び先リージョンとに分割して生成され、この間に結合を持たなければ、リージョンごとの分割が可能となる。

【補足説明】結合は、リンカによる直接的なシンボル参照を前提としている。このため、リージョン分割では、直接的な結合はできない。

2.13.3 複数のシングルトン

リージョン分割する場合、分割されるセルごとに同じセルタイプのシングルトンセルを持つことができる。例えば、カーネルのような資源は、リージョン分割されるリージョンごとに存在するが、それぞれのリージョンにおいてはシングルトンになる。

2.14 分散フレームワークのモデル

TECS における分散フレームワークはスループラグインによって実現される。

2.14.1 リモート呼び出し (Remote Procedure Call, RPC)

リモート呼び出しは、通常の呼び出しとは異なり、呼び先の関数が異なるタスクやプロセッサで処理されるものである。リモート呼び出しを行う目的は、負荷の分散や、呼び先にのみ存在するリソースへのアクセスなどがある。

2.14.2 接続の分類

TECS では、組込みシステムの特性を考慮して、接続を 4 通りに分類する。

- トランスペアレント接続
- オペイク接続
- クローズドネットワーク接続
- オープンネットワーク接続

接続の大きな分類としてトランスペアレント接続とオペイク接続がある。以下に、それぞれの接続の特徴を説明する。

トランスペアレント接続は、呼び元と呼び先でアドレス空間が共通していて、呼び先にポインタが渡された場合、そのポインタを通して呼び元が渡そうとした本来の引数をアクセス可能な状態を言う。トランスペアレント接続は、さらに直接呼び出しとアウトタスク呼び出しがある。直接呼び出しは通常の関数呼び出しであり、結合した状態である。アウトタスク呼び出しは、同一プロセッサの異なるタスクで呼び先の処理が行われる場合が、一般的な例である。メモリ空間を共有していてポインタ値を渡すだけで本来の引数を扱えるのであれば、必ずしも同一プロセッサで処理される必要はない。メモリ対称マルチコアプロセッサの、異なるコアで動作させる場合もトランスペアレント接続となる。

オペイク接続は、メモリ空間は共通しておらず、ポインタが渡されても、そのポインタを通して呼び元が渡そうとした引数をアクセスすることはできない。呼び出しに際して、呼び元が渡そうとしたポインタが指している本来の引数をコピーして渡してやる必要がある。オペイク接続では、クローズドネットワーク接続とオープンネットワーク接続がある。クローズドネットワーク接続では、静的に接続することができるが、オープンネットワーク接続では接続先も接続元も存在性の保証がない。認証などの手続きも必要になる。

2.14.3 リモート呼び出しのモデル

TECS におけるリモート呼び出しは、結合を RPC チャンネルに置き換えることで実現される。RPC チャンネルは実際には RPC チャンネルコンポーネントである。RPC チャンネルは、マーシャラ、アンマーシャラ、チャンネルなどのコンポーネントから構成される。マーシャラ、アンマーシャラはシグニチャごとに異なるものになるため、専用のジェネレータ、すなわちプラグインを用いて生成させる。チャンネルは、組込みシステムにおいては種々のものが想定される。例えば、共有メモリであったり、CAN や専用のリアルタイムネットワークであったりする。このリモート呼び出しのモデルでは、チャンネルとしてのシグニチャを備えセルを用意すればどのようなものであっても置き換えることができる。

2.15 検証支援機能に関するモデル

2.15.1 トレース

呼び口と受け口の間に関数呼び出しをトレースすることで、検証を支援することができる。

トレースは、スループラグインにより実現する。トレースプラグインで、呼び出された関数、引き数、戻り値をログ機構に出力する。

2.15.2 アサート

【制限】アサートについては、未検討である。

第3章 TECS コンポーネント図

3.1 TECS コンポーネント図の位置づけ

コンポーネントの組上げ、すなわちソフトウェア構造の直観的な理解を助けるために、コンポーネント図を用いる。

TECS コンポーネント図は、コンポーネントインスタンスとそれらを組合わせて構築されるシステムの構造を表現するものである。

このため、TECS コンポーネント図は、実装されたソフトウェアの構造と一対一に対応する。具体的な実装を表現するものであり、ハードウェアの回路図に近いものになる。

【補足説明】TECS ではコンポーネント間の関係は、呼び口と受け口の結合のみである。TECS のコンポーネントは(基本的には)静的であり、セルに与えられた名前によりアクセスできる。動的なコンポーネントシステムではコンポーネントへのポインタを動的に引き渡すために結合が曖昧化される可能性があるが、静的なコンポーネントシステムでは曖昧化されることがない。

3.2 TECS コンポーネント図の記述方法

3.2.1 基本的なコンポーネント図

図 3.1 は基本的な TECS コンポーネント図であり、二つのセルが結合された状態を表す。

セルは長方形により表す。長方形の内側に、セルの属するセルタイプのセルタイプ名およびセル名を記す。

受け口は、セルの長方形の辺に沿って塗つぶした三角形を長方形の内向きに置き、受け口名を添える。

呼び口は、セルの長方形の辺から始まる線分により表す。呼び口名を添える。

結合は、呼び口から受け口へ向かう線分により表す。線分に沿ってシグニチャ名を添える。

3.2.2 合流

図 3.2 は結合の合流を表す。一つの受け口に対し、複数の呼び口が結合することはできるが、その逆は許されない。

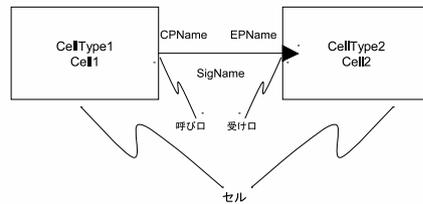


図 3.1: 基本的なコンポーネント図

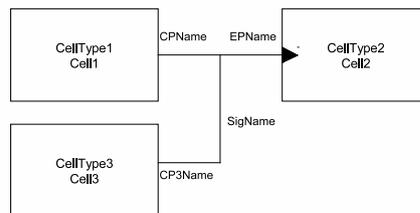


図 3.2: 合流

3.2.3 呼び口配列

図 3.3 に呼び口配列の例を示す。呼び口配列では、呼び口名に配列の添数を添えて記す。

3.2.4 受け口配列

図 3.4 に受け口配列の例を示す。受け口配列では、受け口名に配列の添数を添えて記す。

3.2.5 コールバック

図 3.5 はコールバックの例である。

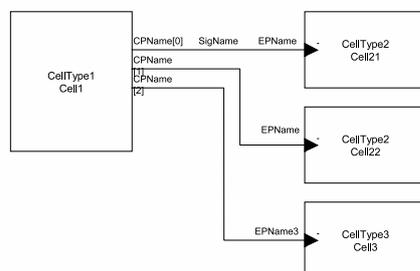


図 3.3: 呼び口配列

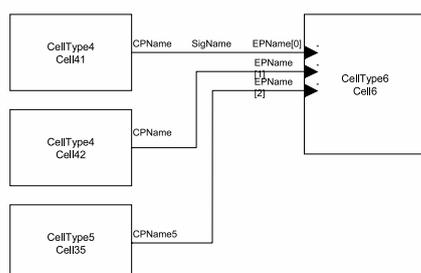


図 3.4: 受け口配列

3.2.6 呼び口配列と受け口配列

図 3.6 は呼び口配列と受け口配列の添数に対応付けて、コールバックを実現した例である。複数の呼び元に対してコールバックを実現する場合には、この例のように呼び口と受け口を対応付けて結合する。

3.2.7 アクティブ

図 3.7 はアクティブセルを表す TECS コンポーネント図である。長方形の辺を二重線により記すことで、アクティブセルであることを示す。

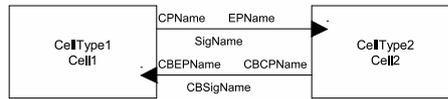


図 3.5: コールバック

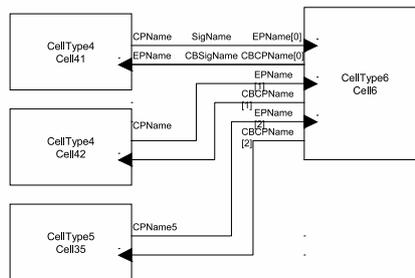


図 3.6: 呼び口配列と受け口配列

3.2.8 複合セル

図 3.8 は複合セル (composite) を示す TECS コンポーネント図である。通常、図 3.9 のように複合セルの内部は描かない。

3.2.9 RPC

図 3.10 はリモート呼び出しを行うための接続を示す TECS コンポーネント図である。通常の結合とは異なり、二重線により接続を表す。この二重線で示される線は、図 3.11 の RPC チャンネルコンポーネントが挿入される。

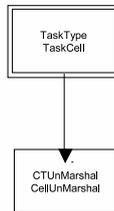


図 3.7: アクティブセル

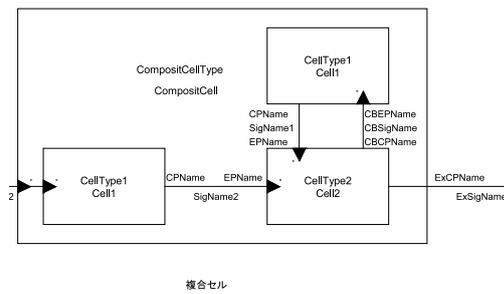


図 3.8: 複合セルの内部構造を示す

3.2.10 アロケータ

アロケータ、以下のように結合の線に近接した からアロケータセルの受け口に結合の線を引いて表す。これは、内部生成されるアロケータ呼び口からアロケータセルへの結合を表す。呼び元と呼び先の両方に呼び口にアロケータ呼び口が内部生成される。

3.2.11 多段リレーモデル

図 3.15 は、多段リレーモデルを表す TECS コンポーネント図である。多段リレーモデルでは、アロケータにより確保されて send 引き数として受け口に渡されたメモリ領域を、再び 呼び口から send 引き数として呼び出す際に引き渡す。

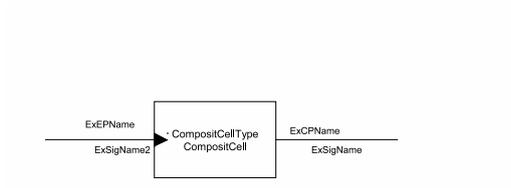


図 3.9: 複合セルの内部構造を見せない

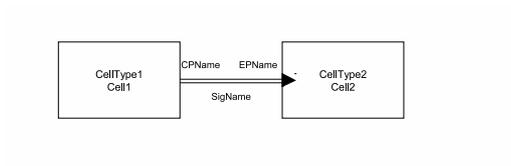


図 3.10: リモート呼び出しの書き方

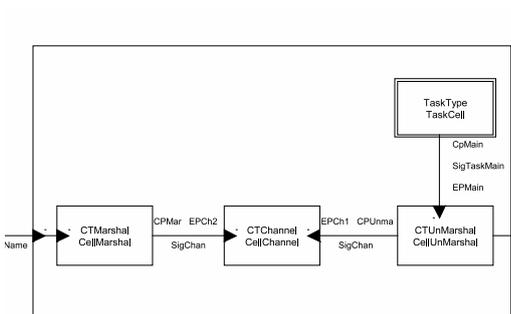


図 3.11: RPC チャンネル

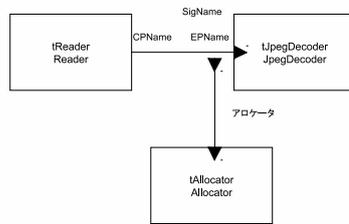


図 3.12: アロケータ

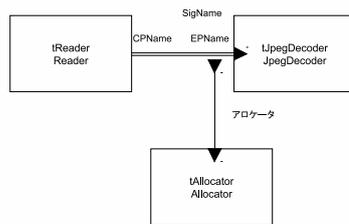


図 3.13: RPC でのアロケータ

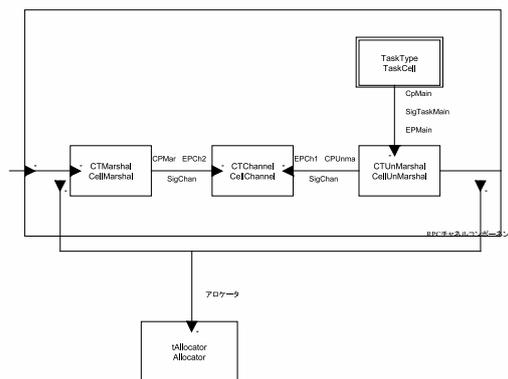


図 3.14: RPC でのアロケータの内部構造

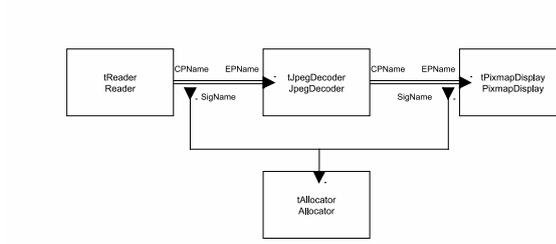


図 3.15: 多段リレーモデル

第4章 TECS コンポーネント記述言語 (TECS CDL)

本章は TECS コンポーネント記述言語 (以下 TECS CDL) を定義する。TECS CDL 言語仕様は、TECS コンポーネントモデルに対する厳密な定義を与える。ただし、コールバックのように TECS の機能としてではなく、使い方として実現されるものは、含まない。

4.1 定義

4.1.1 用語

C 言語と共通部分の用語については、柴田望洋氏後援会のホームページにある「ANSI / ISO / JIS 用語対応表」を参考にしている。

<http://www.bohyoh.com/>

「ANSI / ISO / JIS 用語対応表」

<http://www.bohyoh.com/Books/MeikaiA01/MKA01apnd03.pdf>

4.1.2 文法定義

字句構造、構文構造および構文構造に伴う意味規則により TECS CDL の文法を定義する。

構文構造の定義は Racc の入力ファイルの記法に従う。Racc は LALR(1) 文法のパーサを生成する Ruby で記述されたパーサジェネレータである。以下の URL から情報を得ることができる。

<http://i.loveruby.net/ja/projects/racc/doc/>

【制限】本来ならば 拡張 BNF を定義して、それに基づいて構文定義を行うべきであるが、ここでは簡単に文法を定義するために Racc による記述を構文定義の手段として用いる。

BNF とは、少なくとも以下の点が異なる。

- BNF では ' ::= ' により、左辺と右辺を分けるが、この記法では代わりに ':' を用いる
- 終端記号は単語全体を大文字で記載する
- 記号 (',', '*', 'など) は引用符 "" で囲んで記載する

なお、拡張 BNF に用いられる、(正規表現に基づく) 繰り返しなどの略記法は用いない。

4.1.3 記述単位

TECS CDL によるコンポーネント記述は、補助記憶装置上のファイルに格納する。コンポーネント記述は複数のファイルに分割することができるが、各々のファイルの内容は、構文としては「全記述」である必要がある。ここでの全記述とは、後で説明する、「コンポーネント記述 (component_description)」のことである。

【補足説明】意味的には他のファイルに記述された内容を参照してもよく、構文として「全記述」となればよい。

4.1.4 文字コード

ファイルに使用する文字コードは、少なくとも ASCII コードを包含するものでなくてはならない。

TECS ジェネレータがコンポーネント記述を解釈する際に、文字コードを指定する。

【参照実装における制限】実装は EUC で行うが、EUC, SJIS, UTF8, NONE に対応する。NONE は、すべてを 1 バイトの文字コードとみなして扱う。SJIS を扱う場合には、注意が必要である。文字列リテラル、文字定数リテラルの項を参照。

4.1.5 名前を扱うルール

名前を扱うルールには、以下の 3 つがある。

- 名前有効範囲
- スコープ
- ネームスペース

名前有効範囲は、名前が与えられる対象によって、同じ名前を持つものを宣言できるかを決定する範囲である。

スコープは、名前が与えられたものを、参照可能な範囲である。大域的な名前、ネームスペース内でのみ通用する名前、局所的な名前がある。

ネームスペースは、重複した名前を許すための機構である。

4.1.6 前方参照

TECS CDL では、特別な記述がない限り、前方参照はできない。つまり、名前が与えられたもの（シグニチャ、セルタイプ、型、定数など）は、名前が与えられた場所から以降で参照が可能となる。

【補足説明】ファイルの先頭から末尾に向かって解釈を進める。読み進める方向が前方である。

セルは、前方参照可能である。ただし、プラグインにより生成されるセルを前方参照できるとは限らない。この場合、セルのプロトタイプ宣言を必要とする。プロトタイプ宣言されたセルは、前方参照できる。

セルの結合において、定数を参照する場合、前方参照できる。

4.1.7 宣言と定義

TECS CDL において、あるものの記述が、直接的に変換された形で実行コードに含まれる場合、あるものの記述を定義という。

一方、そうでない場合、宣言と呼ぶ。

例を以下に示す。

「宣言」となるもの

- シグニチャ
- セルタイプ
- 複合セルタイプ
- セルタイプの受け口
- セルタイプの呼び口
- セルタイプの属性
- セルタイプの内部変数
- セルのプロトタイプ

「定義」となるもの

- セル

これからわかるように TECS CDL の記述の内、セル以外、直接的に変換されたコードが実行コードに含まれることはない。

4.1.8 名前・文字列の連結操作

TECS CDL や、特に C 言語に変換する過程で、名前や文字列を連結して、別の名前や文字列に変換する操作が、しばしば用いられる。

操作の対象となる名前や文字列は、'('、')' で囲んで表す。

連結は、'+' で表す。

以下に例を示す。

(ドメインプラグイン名) = (ドメインタイプ名) + 'Plugin'

上記は、ドメインタイプ名に文字列 'Plugin' を連結して、ドメインプラグイン名が生成される。ここで "" は、文字列を区切るもので、文字列には含まれない。

4.2 字句構造

4.2.1 字句

TECS CDL の字句には、以下のものがある。

- キーワード
- 指定子キーワード
- 識別子
- リテラル
- 型名
- 記号

以下にそれぞれの字句について説明する。

4.2.2 空白文字

16 進数で 20 以下の文字コードは空白文字として扱われる。

TECS CDL の字句は 1 文字以上の空白文字によって区切られる。ただし、空白文字がなくても字句が決定できる場合には、この限りではない。

【制限】全角の空白文字は、空白文字とはみなされない。

4.2.3 キーワード

以下の語は、キーワードである。キーワードは、識別子として用いることができない。

```
attr bool_t C_EXP call cell celltype char char_t composite
const Descriptor double64_t entry enum enum16 enum32 enum64
enum8 factory FACTORY false float32_t generate import import_C
int int128_t int16_t int32_t int64_t int8_t long namespace
region require schar_t short signature signed struct true
typedef uchar_t uint128_t uint16_t uint32_t uint64_t uint8_t
unsigned var void volatile
```

4.2.4 指定子キーワード

以下の語は、指定子キーワードである。指定子を記述可能な箇所においてのみキーワードとして扱われる。従って、指定子キーワードを識別子として用いることができる。ただし、指定子キーワードとして扱われる箇所に現れる識別子としては用いることができない。例えば `size_is` などの引数に現れる場合には、指定子キーワードとは異なる語を用いる必要がある。

```
active allocator choice class context count_is deviate
omain dynamic id idx_is_id in in_through inline inout
linkunit node nullable omit oneway optional out
out_through prototype receive ref_desc send singleton
size_is string through to_through
```

4.2.5 識別子

識別子は、アルファベットまたは `'_'` で始まり、アルファベット、数字または `'_'` の繰返しからなる。

TECS CDL において識別子の長さを規定しないが、CDL 記述を C に翻訳する際にネームスペース、セルタイプ名、セル名、呼び口名、受け口名が連結されるため、連結語の長さが C コンパイラの識別可能な長さを超えない配慮が必要になる。

今日の多くのコンパイラは 255 文字までの長さの識別子を識別可能である。少なくとも、この長さに収まるようにすべきである。

4.2.6 リテラル

リテラルには、以下のものがある。

- 16 進定数リテラル (HEX_CONSTANT)
- 8 進定数リテラル (OCTAL_CONSTANT)
- 整数リテラル (INTEGER_CONSTANT)
- 浮動小数リテラル (FLOATING_CONSTANT)
- 文字列リテラル (STRING_LITERAL)
- 山括弧文字列リテラル (AB_STRING_LITERAL)
- 文字定数リテラル (CHARACTER_LITERAL)

16 進定数リテラルは、'0x' で始まり、0-9 および/または a-f, A-F の 1 個以上繰返しである。

8 進定数リテラルは、'0' で始まり、0-7 の 1 個以上繰返しである。

浮動小数リテラルは 1-9 で始まり 0-9 の繰返し (整数部) に続いて '.', 続いて 0-9 の 0 個以上の繰返し (小数部)、さらに続いて 'e' または 'E' に続く、符号 '+' または '-' (省略化)、最後に 0-9 の 1 個以上の繰返し ('e' または 'E' 以降、ここまで指数部) である。小数部、指数部は省略可能である。ただし、0 に続いて '.' が続く場合は、8 進定数リテラルに優先して浮動小数として扱われる。

【補足説明】浮動小数リテラルは C 言語のそれよりも若干制限が強い。例えば ".1" は C 言語では正当な浮動小数であるが、TECS CDL においては浮動小数とはみなされない。

整数定数リテラルは 1-9 で始まり、0-9 の 0 個以上繰返し、または '0' である。

文字列リテラルは、ダブルクォート "" で囲まれた文字列である。文字列リテラルに、ダブルクォートを含めるためには

バックスラッシュ '\'

を前置する。C 言語においては、バックスラッシュを前置することで改行コードなどの制御コードを記述するが TECS CDL においては特に解釈せずバックスラッシュの削除も行わない。これは、文字列リテラルがそのまま C 言語へ渡されるためである。ただし、factory または FACTORY の引数として与えられた場合には別に扱われる。以下は、制御文字コードに置換される。

```
"\n" ... 改行コード
"\r" ... 復帰コード
"\f" ... フィードコード
"\t" ... TAB コード
```

これ以外に、バックslashが前置されている場合、バックslashは取り除かれ、それに続く文字は無条件に文字列の一部として扱われる。

【参照実装における制限】漢字を扱うための文字コードは SJIS, EUC, UTF8 を用いることができる。ただし SJIS を用いる場合には、C コンパイラが SJIS を扱えるものを用いなくてはならない。SJIS の文字コードには、第二バイトに

0x5c ('\' , バックslash)

を含むものがある。SJIS を扱えない C コンパイラでは、第2バイトの 0x5c をエスケープ文字として扱ってしまう。

文字列リテラルが改行文字を含む場合、行末に

バックslash '\'

を置いてエスケープする。行末にエスケープ文字が無いのは、誤りである。文字列リテラルの終わりのダブルクォートを置き忘れた場合、文法エラーとなる箇所は次のダブルクォートが現れる箇所またはファイルの終わりとなる。このため実際のエラー箇所が分りにくくなる。行末にエスケープ文字が置かれていない場合にエラーとし、エラー箇所を分りやすくする。

山括弧文字列リテラルは、山括弧 '[' , ']' で囲まれた文字列である。この文字列には、アルファベットの大文字、小文字、' ', '?', '?' (空白文字) のみを用いることができる。

文字定数リテラルは、シングルクォートで囲まれた一文字である。文字定数リテラルは、その文字コードの整数定数として扱われる。バックslashを前置することで、バックslashに続くもう一文字を文字定数の一部として扱われる。

【参照実装における制限】多バイト文字を一文字として扱うことができる。ただし、C コンパイラが多バイト文字を、文字定数として扱うことができる必要がある。ポータビリティの観点からは、用いることは好ましくない。

4.2.7 型名

typedef により定義される型名は、識別子と同じである。

ただし、typedef により定義されて以降は、「型名」として扱われ識別子とは区別される。このため typedef により型名として定義された識別子は、変数名、関数名の識別子としては用いることができない。

4.2.8 コメント

コメントとして記述された文字列は、TECS CDL の記述として解釈されない。コメントの文字列は、以下の2通りの方法で記述できる。

- `'/*', '*/'` で囲んだ文字列
- `'//'` から行末までの文字列

`'/*'` と `'*/'` の間には、改行文字を含むことができる。しかし、入れ子にすることはできない。

4.3 名前有効範囲

TECS コンポーネントモデルを構成する多くの物に名前が与えられる。名前は、物を識別し、ある物からある物を参照する際に用いられる。名前には、大域的な名前と局所的な名前とがある。局所的な名前は、単一の識別子である。大域的な名前は、ネームスペース識別子である。

名前有効範囲は、単一の識別子によって一意に物を識別できる範囲のことである。名前有効範囲には、属する物により、以下の6種類がある。

- 一般名前有効範囲
- シグニチャ関数名有効範囲
- 仮引数名有効範囲
- セル属性名前有効範囲
- タグ名前有効範囲
- フィールド名前有効範囲

同じ名前有効範囲では、名前として重複した識別子を与えることはできない。同じ名前有効範囲で、異なる物に同一の識別子を名前として与えることを名前の衝突と呼ぶ。ただし一般名前有効範囲においては、名前が有効となる境界を設けることができ、同じ境界の範囲で名前として重複した識別子を与えることはできない。

4.3.1 一般名前有効範囲

以下の物は、一般名前有効範囲に属す。

- シグニチャ
- セルタイプ
- ネームスペース
- リージョン

- セル
- 定数変数
- 型定義

一般名前有効範囲は、全記述の文リストに含まれる物の名前を要素とする名前有効範囲である。一般名前有効範囲は、ネームスペースにより境界を設けることができる。ネームスペースは、入れ子構造をとることができ、内側のネームスペースと外側のネームスペースでは異なる有効範囲となる。

名前の有効範囲（スコープ）は、属するネームスペースおよびその内側のネームスペースである。つまり内側のネームスペースから外側のネームスペースの物は直接参照できる。

この時、内側のネームスペースと外側のネームスペースで重複した識別子を用いられていても名前の衝突は生じないが、内側の物から重複した名前を持つ外側の物を局所的な名前によっては参照することができない。重複した名前を持つ外側の物は「隠蔽」される。

なお、リージョン、セル、、型定義、定数変数は、一般有効名前範囲に属するが、ネームスペースの下に置くことはできない。識別子が重複するのは、ルートネームスペース（ルートリージョンと同じ）においてのみである。

リージョン、セルは、リージョンの下にのみ置くことができる。リージョンは、ネームスペースの一種であり、一般名前有効範囲の境界となる。リージョンは、セルのレイアウトを制御する役目も併せて持つ点でネームスペースと異なる。

4.3.2 シグニチャ関数名名前有効範囲

シグニチャ内の関数名は、シグニチャ関数名名前有効範囲に属し、それぞれのシグニチャにより名前有効範囲が仕切られる。

4.3.3 仮引数名名前有効範囲

関数の仮引数は、仮引数名名前有効範囲に属し、それぞれの関数により名前有効範囲が仕切られる。

4.3.4 セル属性名前有効範囲

以下の物は、セル属性名前有効範囲に属し、それぞれのセルタイプにより名前有効範囲が仕切られる。

- 呼び口名

- 受け口名
- 属性名

4.3.5 タグ名前有効範囲

構造体のタグは、タグ名前有効範囲に属する。

【制限事項】タグ名前有効範囲は、ネームスペースごとに仕切られていない(ネームスペース下に構造体定義を置くことはできない)。

4.3.6 フィールド名前有効範囲

構造体のフィールド名は、フィールド名前有効範囲に属し、それぞれの構造体により名前有効範囲が仕切られる。

4.4 コンポーネント記述

4.4.1 コンポーネント記述

```
component_description
:
| component_description specified_statement
```

コンポーネント記述は、TECS コンポーネント記述言語 (TECS CDL) のコンパイル単位であり、ファイルに記述する目標記号となる。

コンポーネント記述は、指定子文のリストである。

4.4.2 指定子文

```
specified_statement
: statement
| '[' statement_specifier_list ']' statement
```

指定子文は、文または、'[', ']' で囲まれた文指定子リスト続く文である。

4.4.3 文

```
statement
  : import
  | import_C
  | typedef
  | namespace
  | signature
  | celltype
  | cell
  | composite_celltype
  | enum_specifier ';'
  | struct_specifier ';'
  | const_statement
  | region
  | signature_plugin
```

コンポーネント記述の文は、以下のいずれかである。

- インポート (import) 文
- C 言語インポート (import_C) 文
- 型定義 (typedef) 文
- ネームスペース (namespace) 文
- シグニチャ(signature) 文
- セルタイプ (celltype) 文
- セル (cell) 文
- 複合セルタイプ (composite) 文
- 列挙指定子
- 構造体指示子
- 定数定義文
- リージョン (region) 文
- シグニチャプラグイン (generate) 文

コンポーネント記述に現れるセル文およびリージョン文を「組上げ記述」と呼ぶ。定数定義文は、構文要素としては宣言文となる。

4.4.4 文リスト

```
statement_list
    : statement
    | statement_list statement
```

文は、文のリストである。

4.5 インポート (import) 文

4.5.1 インポート (import) 文

```
import
: 'import' '(' STRING_LITERAL ')' ';'
| 'import' '(' AB_STRING_LITERAL ')' ';' ;
```

インポート文は、別のファイルに分割して記述されたコンポーネント記述を取込むためのものである。

インポート文は文字列リテラルまたは山括弧文字列リテラルを引数にとり、文字列リテラルまたは山括弧文字列リテラルで指定されるコンポーネント記述ファイルを、読み込み中のコンポーネント記述ファイルの一部として読み込む。ただし、既にインポートされているコンポーネント記述ファイルは、読み込まない。

読み込みはインポート文が現れたところで行われるため、読み込まれたコンポーネント記述は、インポート文よりも後に記述される文から参照できる。

読み込まれるコンポーネント記述ファイルは、構文的に閉じていなければならない。例えば、`cell` や `celltype` などの記述を途中で終わって、残りの部分は別のコンポーネント記述ファイルからインポートするようなことはできない。

しかし、意味的に閉じている必要はない。例えば2つのインポート文が記述されている場合、1つ目にインポートされるコンポーネント記述ファイルで定義されたものを2つ目のコンポーネント記述ファイルで参照することはできる。

インポート文により読み込まれるコンポーネント記述ファイルの中にインポート文を記述することができる。つまり再帰的なインポートは可能である。再帰的な読み込みの多重度の制限は実装依存である。

コンポーネント記述ファイルが、山括弧文字列リテラルで指定された場合、そのコンポーネント記述が再利用されていることを表す。再利用されているコンポーネント記述ファイルから再帰的にインポートされるコンポーネント記述ファイルは、再利用されているとみなされる。ただし、同じコンポーネント記述ファイルが複数回インポートされる場合、最初にインポートされた時の再利用指定が有効である。

【補足説明】同じコンポーネント記述ファイルが複数回インポートされる場合で、再利用指定が最初にインポートされる時と異なる場合、TECS ジェネレータが警告するのが望ましい。

再利用されているコンポーネント記述ファイルにセルタイプ定義が現れた場合、そのセルタイプが再利用されているものとみなされる。

【補足説明】コンポーネント記述が再利用されている場合、それに含まれるセルタイプを開発済みとみなし、TECS ジェネレータはテンプレートファイルを生成しない。

4.6 C 言語インポート (import_C) 文

4.6.1 C 言語インポート (import_C) 文

```
import_C
    : IMPORT_C '(' STRING_LITERAL ')' ';'
    | IMPORT_C '(' STRING_LITERAL ',' STRING_LITERAL ')' ';' ;
```

C 言語インポート文は、C 言語で記述された型定義を取込むためのものである。型定義以外の記述があっても、それらは読み飛ばされて無視される。

C 言語インポート文は文字列リテラルを一つ、ないし二つ引数にとり、第一引数で指定される C 言語ヘッダファイルに含まれる型定義、構造体定義、または列挙型定義を取込む。

上記以外の定義、宣言は、取込まれない。すなわち、マクロ定義、変数定義などは取込まれない。TECS CDL で、これらを参照するための手段として C_EXP がある (C_EXP については初期化子の項を参照)。

第二引数はオプションで、マクロ定義 (プリプロセッサの -D オプション) を文字列として指定する。複数のマクロ定義を行う場合には ';' で区切って指定する。デフォルトで TECS が定義される。

以下に例を示す。2 つ目の例は `inline` を無視するように指定されている。

```
import_C( "myheader.h" );
import_C( "kernel/kernel.h", "TECS_KERNEL,_inline=" );
```

【使用上の注意】TECS の組込み型、すなわち `int8.t`, `int16.t` など型指定子として定義されたものの C 言語における型を定義したヘッダファイルを、必ず `import_C` により取込む必要がある。取込まなくても TECS CDL の解釈には成功するが、TECS ジェネレータの生成したプログラムをコンパイルする際にエラーとなる。

【補足説明】TOPPERS/ASP 環境においては、`kernel.h` を `import_C` する。

【参照実装における制限】列挙型は取込まれない。

【参照実装における制限】現在の参照実装では、すべての正当な C 言語記述を処理できるわけではない。

【参照実装における制限】現在の参照実装では、`inline`, `__inline__` をキーワードとして扱う。読み飛ばし可能なキーワードは `-D _inline=` などのように空文字列に定義するか、上記の例のように `import_C` の引数でマクロ定義することで無視する。

4.6.2 重複インクルード

`import_C` は、以前に読み込まれた `import_C` 文の第一引数で指定されたものと同じ名前の C 言語ヘッダファイルが指定された場合、重複してインポートしない。ただし、この場合、第二引数は、第二引数が指定されない場合を含めて、同じでなければならない。第二引数が異なれば、エラーとする。

しかし、`import_C` によって読みこまれるヘッダファイルの中からインクルードされる他のヘッダファイルについて、重複インクルードされる。

【参照実装における制限】重複インクルードされた結果、同じ型の `typedef` が繰り返し行われた場合、重複エラーではなく文法エラーとなる。

【仕様決定の理由】`import_C` は `#define` を次の `import_C` に引き継がないためである。

4.7 文指定子

文指定子は、文に前置して、文に指示を与える。文指定子は、文に前置するものとして、一まとまりとして扱うが、実際には、ある文指定子が前置できる文は限られている。

4.7.1 文指定子リスト

```
statement_specifier_list
    : statement_specifier
    | statement_specifier_list ',' statement_specifier
```

文指定子リストは、文指定子または、文指定子を ',' で連結したものである。

4.7.2 文指定子

```
statement_specifier
    : 'context' '(' string_literal_list ')' # signature
    | 'deviate' # signature
```

```

| 'active'                # celltype, composite
| 'idx_is_id'            # celltype, composite (composite: no-effect)
| 'singleton'            # celltype, composite
| 'allocator' '( alloc_list )' # cell
| 'id' '( constant_expression )' # cell
| 'prototype'            # cell
| 'generate' '( plugin_name ',' plugin_arg )' # celltype, cell

```

【補足説明】構文上、文指定子は、どの文にも指定できる。共通のものであるため、構文規則はまとめられている。

文指定子は、指定できる対象が限定されている。

1) シグニチャに指定できるもの

- コンテキスト指定子 (context)
- 逸脱指定子 (deviate)

2) セルタイプ、複合セルタイプに指定できるもの

- アクティブ指定子 (active)
- idx_is_id 指定子 (idx_is_id)
- シングルトン指定子 (singleton)

3) セルに指定できるもの

- アロケータ指定子 (allocator)
- id 指定子 (id)
- プロトタイプ指定子 (prototype)

4) セルタイプとセルに指定できるもの

- ジェネレート指定子 (generate)

4.7.3 コンテキスト指定子 (context)

コンテキスト指定子は、引数としてコンテキスト名文字列を取る。コンテキスト指定子は、シグニチャにのみ指定できる。また、一つのシグニチャに対し、一回だけ指定できる。

コンテキスト名文字列は、通常以下のいずれかである。

- "task" ... タスク部
- "non-task" ... 非タスク部

- "any" ... いずれのコンテキストも可能

コンテキストが指定されていないシグニチャは、"task" が指定されたものと仮定される。

【補足説明】コンテキストの指定は、コンポーネントの設計者、および利用者に対するメモである。TECS ジェネレータは、この記述に基く検査はしない。

【補足説明】assert によるコンテキスト検証を実施する場合には、コードが自動生成される（予定である）。

4.7.4 逸脱指定子 (deviate)

逸脱指定子は、シグニチャにのみ指定できる。

逸脱指定子は、シグニチャの関数において、逸脱とみなされる引き数が指定されていることを表す。

4.7.5 アクティブ指定子 (active)

アクティブ指定子は、セルタイプがアクティブであることを示す。アクティブなセルタイプとは、OS の資源であるタスクを内包し、受け口関数が呼び出されない場合であってもセルが動作する（セルタイプコードが実行される）。

アクティブなセルタイプではなく、受け口を持たない場合、以下のいずれかを満たす場合、誤りである。

- シングルトンセルタイプでなく、かつセルのファクトリを持たない
- シングルトンセルタイプであって、セルタイプのファクトリもセルのファクトリも持たない

【補足説明】TECS ジェネレータは、上記の誤りを検出してもエラーとはせず、警告するに留める。

アクティブセルタイプに属するセルの、どの受け口に結合がない場合にも、上記のいずれかを満たす場合、誤りである。

【補足説明】TECS ジェネレータは、上記の誤りを検出してもエラーとはせず、警告するに留める。

複合セルタイプにおいて、内部セルのセルタイプがアクティブであるものが一つでもある場合、その複合セルタイプは、必ずアクティブ指定子が指定されなくてはならない。

複合セルタイプにおいて、内部セルのセルタイプがアクティブであるものが一つもない場合、その複合セルタイプに、アクティブ指定子を指定してはならない。

4.7.6 idx_is_id 指定子 (idx_is_id)

idx_is_id 指定子は、セルの実行時識別として整数値を用いることを示す。この指定がない場合、実行時識別としてポインタが用いられる。実行時識別として整数値を用いることの利点は、指定値の有効性検査が行われることである。

4.7.7 シングルトン指定子 (singleton)

シングルトン指定子は、セルタイプのセルがシステム中に一つだけ存在することを示す。ただし、別にコード生成されるリージョンにおいて、あるシングルトンセルタイプのセルが最大1つだけ存在することはできる。つまり、あるコンポーネント記述において、あるシングルセルトンタイプのセルが複数存在することができる。

【補足説明】例えば分散している場合の OS カーネルを想定したものである。

複合セルタイプにおいて内部セルのセルタイプがシングルトンであるものが一つでもある場合、その複合セルタイプは、必ずシングルトン指定子が指定されなくてはならない。

複合セルタイプにおいて内部セルのセルタイプがシングルトンであるものがない場合であっても、その複合セルタイプに、シングルトン指定子することができる。

4.7.8 アロケータ指定子 (allocator)

アロケータ指定子は、引数として alloc のリストを取る。

```
alloc_list
    : alloc
    | alloc_list ',' alloc

alloc
    : IDENTIFIER '.' IDENTIFIER '.' IDENTIFIER '=' initializer
    | IDENTIFIER '[' constant_expression ']' '.' IDENTIFIER
        '.' IDENTIFIER '=' initializer
```

alloc は、(1) 3つの識別子を '.' で連結し、 '=' の右辺に初期化子を置いたものである。または (2) 最初の識別子に続いて '['、']' で囲まれた定数式、続いて '.', 識別子, '.' 識別子, '=', 初期化子を置いたものである。

(1) の場合、最初の識別子はアロケータを指定するセルの受け口の名前、2番目の識別子は受け口の関数名 (send または receive 指定子された引数を持つ)、3番目は仮引数名である。

(2) は呼び口配列の場合で、'[', ']' で囲まれた定数式で、配列添数を指定する。

セルのプロトタイプ宣言が、セルの定義に先立ってなされる場合、アロケータ指定子は、そのプロトタイプ宣言でなされる必要がある。呼び元では呼び先に指定されたアロケータセルを参照するが、受け側のセルが定義される時点で受け側のアロケータが指定されていないとてはならない。

【補足説明】アロケータは、個別に指定する必要がある。

【未決定事項】まとめて指定する手段として、ワイルドカードで指定する手段が予約されている。

4.7.9 id 指定子 (id)

id 指定子は、セルの id 番号を指定するのに用いられる。

詳細は、セルの id 番号の項で説明する。

4.7.10 プロトタイプ指定子 (prototype)

プロトタイプ指定子は、', ' で囲まれた結合リストを伴うセル文が、プロトタイプ宣言であることを示す。

詳細は、セルの項で説明する。

4.7.11 プラグイン指定子 (generate)

プラグイン指定子は、セルタイプ、またはセルにプラグインを適用を指示するものである。プラグイン指定子は、引数を2つ取る。

第一引数は、プラグイン名 (識別子) である。

第二引数は、プラグイン引数 (文字列リテラル) である。

セルタイプにはセルタイププラグイン、セルにはセルプラグインを適用する。

セルタイププラグインは、TECS CDL あるいは TECS ジェネレータの機能では実現できないものを、補完する目的で用いることができる。

セルプラグインは、セルの結合を追加したり、別のセルを生成させるために用いることができる。

4.8 型

4.8.1 型指定子修飾子リスト

```

type_specifier_qualifier_list
    : type_specifier
    | type_qualifier type_specifier_qualifier_list

```

4.8.2 型指定子

```

type_specifier
    : 'void'                # void 型
    | 'float32_t'           # IEEE 754 単精度 (32bit) 浮動小数点
    | 'double64_t'         # IEEE 754 倍精度 (64bit) 浮動小数点
    | struct_specifier     # 構造体型
    | enum_specifier       # 列挙型
    | TYPE_NAME            # 型名
    | sign_int_type        # 符号付整数型
    | char_type            # 文字型
    | 'bool_t'            # ブール型

```

【使用上の注意】以下は、使用できる、推奨されない。

```

    | 'bool'
    | 'float'
    | 'double'

```

4.8.3 文字型

```

char_type
    : 'char_t'             # char と同じです
    : 'schar_t'           # 有符号 8 bit 整数
    : 'uchar_t'           # 無符号 8 bit 整数

```

`char_t` は、実装によっては 8bit を超える精度を持つが 8bit のみ有効として扱われることを仮定します。従って、RPC では 8 bit のみ受渡しされます。

4.8.4 整数型

```

int_type
  : 'char'          # サイズは実装依存．可能な限り使用を避ける
べきです
  : 'short'         # サイズは実装依存．可能な限り使用を避ける
べきです
  | 'int'           # サイズは実装依存．可能な限り使用を避ける
べきです
  | 'long'          # サイズは実装依存．可能な限り使用を避ける
べきです
  | 'int8_t'        # 有符号 8 bit
  | 'int16_t'       # 有符号 16 bit
  | 'int32_t'       # 有符号 32 bit
  | 'int64_t'       # 有符号 64 bit
  | 'int128_t'      # 有符号 128 bit
  | 'uint8_t'       # 無符号 8 bit
  | 'uint16_t'      # 無符号 16 bit
  | 'uint32_t'      # 無符号 32 bit
  | 'uint64_t'      # 無符号 64 bit
  | 'uint128_t'     # 無符号 128 bit

```

【補足説明】以下は、初期の仕様で使われ V1.3.0.1 まで互換性のため使用可能であった。

```

  | 'int8'
  | 'int16'
  | 'int32'
  | 'int64'
  | 'int128'

```

4.8.5 符号

```

sign
  : 'signed'        # 有符号
  | 'unsigned'      # 無符号

```

4.8.6 符号付整数型

```

sign_int_type
  : sign int_type

```

```
| int_type
```

uintN_t (N:8,16,32,64,128) に signed を付加することはできない。

【使用上の注意】 signed intN_t と unsigned uintN_t は旧仕様との互換性のため使用できますが、使わないようにしてください。

4.8.7 型修飾子

```
type_qualifier
: 'const'
| 'volatile'
```

4.8.8 型名

キャスト演算子として使用する。

```
type_name
: type_specifier_qualifier_list
| type_specifier_qualifier_list abstract_declarator
```

4.9 構造体

4.9.1 構造体指示子

```
struct_specifier
: 'struct' struct_tag '{' struct_declaration_list '}'
| 'struct' '{' struct_declaration_list '}'
| 'struct' struct_tag
```

【補足説明】 K&R の struct_or_union_specifier に相当するが、union は使えない【未決定事項】 struct tag の namespace 対応

4.9.2 構造体宣言リスト

```
struct_declaration_list
: struct_declaration
| struct_declaration_list struct_declaration
```

4.9.3 構造体タグ

```
struct_tag:
IDENTIFIER
```

4.9.4 構造体宣言

```
struct_declaration
: type_specifier_qualifier_list struct_declarator_list ';'
| '[' pointer_specifier_list ']'
  type_specifier_qualifier_list struct_declarator_list ';' ;
```

【参照実装における制限】構造体メンバのポインタ指定子に対応していない

4.9.5 構造体宣言子リスト

```
struct_declarator_list
: struct_declarator
| struct_declarator_list ',' struct_declarator
```

4.9.6 構造体宣言子

```
struct_declarator
: declarator
```

構造体宣言子は、構造体のメンバー変数を宣言するのである（関数であってはならない）。可変長配列は、ポインタ型に `size_is`, `count_is`, `string` 指定子を指定する。添数を省略した配列をメンバー変数とすることはできない。

【補足説明】ビットフィールドは使えない

4.10 ポインタ

4.10.1 ポインタ型

```
pointer
: '*'
| '*' type_qualifier
| '*' pointer
| '*' type_qualifier pointer
```

4.10.2 ポインタ指定子リスト

```
pointer_specifier_list
    : pointer_specifier
    | pointer_specifier_list ',' pointer_specifier
```

4.10.3 ポインタ指定子

```
pointer_specifier
    : 'string'
    | 'string' '(' expression ')
    | 'size_is' '(' expression ')
    | 'count_is' '(' expression ')
```

4.11 列挙

【参照実装における制限】列挙は実装されない

4.11.1 列挙指定子

```
enum_specifier
    : enum_type '{' enumerator_list '}'
    | enum_type IDENTIFIER '{' enumerator_list '}'
    | enum_type IDENTIFIER
```

4.11.2 列挙型

```
enum_type
    : 'enum'
    | 'enum8'
    | 'enum16'
    | 'enum32'
    | 'enum64'
    | 'enum128'
```

4.11.3 列挙子リスト

```
enumerator_list
    : enumerator
    | enumerator_list ',' enumerator
```

4.11.4 列挙子

```

enumerator
  : IDENTIFIER
  | IDENTIFIER '=' constant_expression

```

4.12 宣言子

宣言子は、前置される `type_specifier_qualifier_list` を伴って、変数の定義、関数ヘッダの定義に用いられる。

4.12.1 宣言子

```

declarator
  : pointer direct_declarator
  | direct_declarator

```

宣言子は、ポインタ型に続く直接宣言子、または直接宣言子である。

ポインタ型は、後続の直接宣言子の示すものがポインタ型であることを表す。直接宣言子により、変数か、配列か、あるいは関数が示される。

4.12.2 直接宣言子

```

direct_declarator
  : IDENTIFIER
  | '(' declarator ')'
  | direct_declarator '[' constant_expression ']'
  | direct_declarator '[' ']'
  | direct_declarator '(' parameter_type_list ')'
  | direct_declarator '(' ')'

```

直接宣言子は、変数、関数、配列を宣言する場合に用いる。いずれの場合にも、直接宣言子には一つだけ識別子が含まれる。ただし、定数式やパラメータタイプリストに含まれるものを除く。

変数を宣言する場合には `IDENTIFIER` のみの直接宣言子を用いる。

一次元の配列型の変数を宣言する場合には `IDENTIFIER` に続いて `'[expression]'` を置く。N次元の配列型の変数を宣言する場合には (N-1)次元の配列型の直接宣言子に続いて `'[expression]'` を置く。

【制限】添数がない `'[]'` は、ポインタ型を表すものとはみなされない。通常使用しない。

関数型を宣言する場合には IDENTIFIER に続いて '(' parameter_type_list ')' を置く。

関数へのポインタ型を宣言する場合には '(' declarator ')' に続いて '(' parameter_type_list ')' が置く。この場合 declarator はポインタ型でなくてはならない。

上記以外の直接宣言子は、誤りである。

4.12.3 宣言子リスト

```
declarator_list
    : declarator
    | declarator_list ',' declarator
```

宣言子リストは、宣言子を ',' を区切り文字としてつないだリストである。

4.13 パラメータ

パラメータは、関数ヘッダの仮引き数の定義に用いられる。

4.13.1 パラメータタイプリスト

```
parameter_type_list
    : parameter_list
    | parameter_list ',' '...'
```

パラメータタイプリストは、パラメータリストである。

逸脱であるが、パラメータリストの末尾に ',', '...' を置くことができ、可変個数のパラメータを持つことを示す。

【参照実装における制限】可変個数のパラメータ未対応

4.13.2 パラメータリスト

```
parameter_list
    : parameter_declaration
    | parameter_list ',' parameter_declaration
```

パラメータリストは、パラメータ宣言を ',' を区切り文字として連結したリストである。

4.13.3 パラメータ宣言

```
parameter_declaration
    : '[' parameter_specifier_list ']' declaration_specifiers declarator
# 以下はエラー
    | declaration_specifiers declarator
    | declaration_specifiers
    | '[' parameter_specifier_list ']' declaration_specifiers
```

パラメータ宣言は、'[', ']' によって囲まれたパラメータ指定子リスト、宣言指定子、宣言子からなる。パラメータ指定子リストと宣言子は省略できない。

4.13.4 パラメータ指定子リスト

```
parameter_specifier_list
    : parameter_specifier
    | parameter_specifier_list ',' parameter_specifier
```

パラメータ指定子リストは、パラメータ指定子を','を区切り文字として連結したリストである。パラメータ指定子は、同じパラメータ指定子リストの中に、同時には指定できない組合せがある。また、パラメータ指定子によってパラメータの型が限定される。

4.13.5 パラメータ指定子

```
parameter_specifier
    : 'in'
    | 'out'
    | 'inout'
    | 'send'      '(' IDENTIFIER ') '
    | 'receive'   '(' IDENTIFIER ') '
    | 'string'
    | 'string'    '(' expression ') '
    | 'size_is'   '(' expression ') '
    | 'count_is' '(' expression ') '
    | 'nullable'
```

パラメータ指定子には、パラメータの入出力方向を決定付ける基本指定子と、それ以外、すなわち型を修飾する補助指定子とがある。

基本指定子は、キーワード "in", "out", "inout", "send", "receive" のいずれかである。基本指定子は、どれか一つだけ、必ず指定されなければならない。

`in`, `send` は、呼び出される関数に対する入力であることを表す。`out`, `receive` は、呼び出される関数からの出力であることを表す。`inout` は、`in` と同時に `out` も行うことを表す。この場合、`out` にゆるされた型のみパラメータの型として取ることができる。`send`, `receive` は、引数としてアロケータのシグニチャの識別子を取り、入力または出力する値がアロケータセルによって確保されたメモリ領域を介して行われることを表す。

基本指定子の他に、ポインタ型を修飾する `size_is`, `count_is`, `string` がある。

【参照実装における制限】 `[out,size_is(len)]int32_t param0[]` のように `[]` はポインタを表さない (添数有でも同じ)

4.13.6 in 基本指定子

`in` は、パラメータが呼び出される関数に対する入力であることを示す。`in` が指定されている場合、パラメータの型は非ポインタ型か、単一のポインタ型または、`size_is(*1)` と `string` が指定された二重ポインタ型でなくてはならない。

これ以外のパラメータの型は、逸脱として扱われる。逸脱した型は、TECS CDL の範囲では厳密なデータ構造が定義されない。曖昧のない型定義は TECS の一つの目標であり動機ともなっており、TECS の利用者は、逸脱を極力避けなければならない。

パラメータの型がポインタ型の場合、`const` 型修飾子が指定されていなくてはならない。

ポインタ型または配列型の定義例を以下に示す。非ポインタ型、非配列型の場合は、制限がないため、例を省略する。

`oneway` 関数で、`in` 基本指定子が指定されたパラメータにより、自動変数上の配列や構造体へのポインタを渡すのは誤りである可能性が高く、通常避けるべきである。呼び先の関数よりも、呼び元の関数の方が先に終了することが想定されるためであり、呼び元の自動変数上のメモリ領域が、呼び先の関数が実行途中であるにも関わらず、解放されてしまう可能性がある。

1) 文字列

```
[in,string]const int16_t *wstr
```

`wstr` は NULL 終端されている。

2) 次元配列

1. 1) `int32_t` の大きさ 4 の配列

```
[in,size_is(4)]const int32_t *array
```

ポインタが配列であることを表す。

1. 2) int32_t の大きさ 4 の配列 (推奨しない記法)

```
[in]          const int32_t  array[4]
```

この記述は 2.1) と同じに扱われる

1. 3) int32_t の大きさ 4 の配列へのポインタ

```
[in]          const int32_t  (*array)[4]
```

この記述も 2.1) と同じであるが、ポインタ自体は配列として扱われていない。

1. 4) ポインタの配列 (逸脱)

```
[in]          const int32_t *array[4]
```

ポインタの配列であり、逸脱である。配列要素のポインタが、配列かどうか、配列であれば大きさが客観的にはわからない。

【補足説明】ポインタが必ず非配列を指すとすれば、逸脱でなく扱えるはずだが、逸脱とする。

3) 文字列の配列

```
[in,size_is(sz),string]const int16_t **str
```

1. 1) 文字列の配列 (逸脱)

```
[in,size_is(sz),string]const int8_t ***str
```

これは逸脱である。size.is は str に最も近いポインタが配列を指し、配列の大きさが別のパラメータ sz の値であることを表す。string は int8_t に最も近いポインタが NULL 終端された int8_t の配列を指すことを示す。中央のポインタについては、ポインタの指すものが配列か非配列か、配列の大きさ、あるいは NULL 終端された配列かどうか指定されず、曖昧になるため逸脱となる。

4) 二次元配列

1. 1) 4 × 4 の int64_t の 2 次元配列

```
[in,size_is(4)]const int64_t  (*array2D)[4]
```

1. 2) 4 × 4 の int64_t の 2 次元配列へのポインタ

```
[in]          const int64_t  (*array2D)[4][4]
```

【補足説明】ポインタ型に const 型修飾子が指定されていなくてはならないが、以下の問題が残っている。

```

typedef int *PType;
typedef const int *CType;
signature sSig {
    ER func( [in]const PType ptr );    <<< エラー (*)
    ER func2( [in]CType ptr );        <<< OK
};

```

(*)ptr が const であるが、ptr の指す先は const にならない。TECS ジェネレータは "ptr must be const for 'in' parameter PtrType" というエラーを表示する。

4.13.7 send 基本指定子

send は、パラメータが呼び出される関数に対する入力で、アロケータによって割り付けられたメモリ空間を指すポインタを呼び出される関数に渡すことを示す。呼び出された関数は、渡されたポインタによって示されるメモリ領域を解放しなくてはならない。send 指定されたパラメータの型は単一ポインタ型あるいは、string 指定された二重ポインタ型でなくてはならない。これら以外のポインタ型は、逸脱として扱われる。

ポインタが指す先に出現するポインタ型のデータについても send で指定されたアロケータによりアロケートされるものとする。この規則に従わない場合も逸脱となる。

send はアロケータのシグニチャを引数に取る。ここではアロケータのシグニチャのみを指定し、具体的なアロケータセルは、組上げ記述において指定する。アロケータセルによりアロケートして得られたポインタは、同じアロケータセルによってデアロケートしなくてはならない。

```

ex)
    param0=999
    param1=param0

```

in のように const を指定する必要はない。呼び元は、呼び出し後、メモリ領域を参照することはできない。

以下に、複雑な型の例を示す。

1) 文字列の配列

```
[send(sAllocator),size_is(len),string]char_t **str_array
```

ポインタは sAllocator をシグニチャとするアロケータセルによって割り付けられたメモリ領域を指す。

2) 構造体

```

struct finfo {
    [string]char_t *name;
    size_t      size;
    date_t      date;
};

[send(sAllocator),size_is(len)]struct finfo *finfo

```

構造体のポインタ型メンバ変数についても、アロケータによってアロケートされているものとする。

【参照実装における制限】構造体のポインタ型メンバ変数は未対応。

1. 1) 構造体 (非ポインタ)

```
[send(sAllocator)]struct finfo finfo
```

パラメータが非ポインタの構造体であっても、ポインタ型のメンバ変数についてはアロケータによりアロケートされるものとする。

4.13.8 out 基本指定子

out は、呼び出される関数からポインタの指す先のメモリ領域にデータが出力されることを示す。従って、パラメータは、出力されるデータを格納するメモリ領域を指す単一ポインタ型または、size_is(*1) および string 指定された二重ポインタ型でなくてはならない。これ以外のパラメータの型は、逸脱として扱われる。

出力されるデータが格納されるメモリ領域は呼び元の関数が準備しなくてはならない。out の場合、関数呼び出しで実際に渡される値は、in のポインタ型を指定する場合と同様で、値をやり取りするためのメモリ領域へのポインタを渡す。in の場合は、呼び元の関数がメモリ領域に書込んだ値を呼び先の関数が読み出すが、out の場合には、反対に、呼び先がメモリ領域に書込んだ値を呼び元関数が読み出す。

長さ引数のない string 指定されたパラメータの場合、呼び元がどれだけの長さの文字列が返されるか曖昧になる。このため string 指定子には長さ引数がなくてはならない。

【補足説明】out の string 指定子に引数がない場合は逸脱とする。バッファオーバーラン不具合に繋がる可能性がある。

【補足説明】string の場合には、大きさではなく、長さと言うが、両者に違いはない。

ポインタ型または配列型の定義例を以下に示す。非ポインタ型の場合は、指定できない。

- 1) 文字列 (wstr は NULL 終端されている)

```
[out,string(16)]int16_t *wstr
```

最大長さが定数指定された文字列。呼び出す側は `wstr` に値を返すべきメモリ領域へのポインタを渡す。

2) 一次元配列

1. 1) `int32_t` の大きさ 4 の配列

```
[out,size_is(4)]int32_t *array
```

ポインタが配列であることを表す。

1. 2) `int32_t` の大きさ 4 の配列 (推奨しない記法)

```
[out]          int32_t array[4]
```

この記述は 2.1) と同じに扱われる

1. 3) `int32_t` の大きさ 4 の配列へのポインタ

```
[out]          int32_t (*array)[4]
```

この記述も 2.1) と同じであるが、ポインタ自体は配列として扱われていない。

1. 4) ポインタの配列 (逸脱)

```
[out]          int32_t *array[4]
```

ポインタの配列であり、逸脱である。配列要素のポインタが、配列を指すのか非配列を指すのかどうか、配列を指すのでであれば大きさが客観的にはわからない。

【補足説明】ポインタが必ず非配列を指すとすれば、逸脱でなく扱えるはずだが、逸脱とする。

3) 文字列の配列

```
[out,size_is(sz),string(64)]int16_t **str
```

1. 1) 文字列へのポインタの配列 (逸脱)

```
[out,size_is(sz),string(96)]int8_t ***str
```

これは逸脱である。`size_is` は `str` に最も近いポインタが配列を指し、配列の大きさが `sz` であることを示す。`string` は `int8_t` に最も近いポインタが `NULL` 終端された `int8_t` の配列を指すことを示す。中央のポインタについては、ポインタの指すものが配列か非配列か、配列の大きさ、あるいは `NULL` 終端された配列かどうか指定されず、曖昧であり逸脱となる。

4) 二次元配列

1. 1) 4×4 の `int64_t` の 2 次元配列

```
[out,size_is(4)]int64_t (*array2D)[4]
```

1. 2) 4×4 の `int64_t` の 2 次元配列へのポインタ

```
[out] int64_t (*array2D)[4][4]
```

4.13.9 receive 基本指定子

`receive` は、呼び先の関数からデータが出力されることを示すが、データは呼び先の関数がアロケータによって確保した領域に置かれる。このためパラメータは、呼び先が確保したメモリ領域のポインタを格納するためのメモリ領域へのポインタとなる。呼び元は、ポインタを格納するためのメモリ領域へのポインタ (ポインタ型の変数へのポインタ) を呼び先に渡す。つまり、パラメータは二重ポインタ型でなくてはならない。あるいは `size_is(*1)` と `string` が指定されている場合には三重ポインタ型になる。

ポインタが指す先に出現するポインタ型が指す先のデータについても `receive` で指定されたアロケータによりアロケートされるものとする。

アロケートに失敗して `NULL` を返すことが想定される場合 `nullable` 指定子も併せて指定する。

以下に例を示す。

- 1) 構造体

```
struct complex_number {
    double64_t real;
    double64_t imaginal;
};
```

```
[receive(sAllocator)]struct complex_number **dat
```

構造体へのポインタを返す。

1. 1) 構造体配列

```
[receive(sAllocator),size_is(4)]struct complex_number **dat
```

構造体の配列へのポインタを返す。

- 2) 整数配列

```
[receive(sAllocator),size_is(sz)]int32_t **array
```

大きさ `sz` の配列へのポインタを返す。

- 3) 文字列

```
[receive(sAllocator), string] char_t          **str
```

文字列へのポインタを返す。out の場合と異なり string に大きさ指定がなくても逸脱とはならない。

4) 文字列配列

```
[receive(sAllocator), size_is(5), string] char_t  ***str
```

文字列へのポインタの配列を返す。

4.13.10 size_is, count_is, string 指定子

string, size_is, string 指定子は、ポインタ型を修飾し、ポインタが配列を指すことを表すとともに、その配列の大きさを指定する。これらの指定のないポインタ型は、非配列である。

4.13.11 size_is 指定子と count_is 指定子の特性

- size is と count is は一対で、ポインタ型が配列を指すことを表す。
- size is と count is はパラメータに最も近いポインタ型を修飾する。ただし基本指定子が receive の場合は、パラメータから 2 番目に近いポインタ型を修飾する
- size is, count is は引数を取るが、引数は定数式、または同じパラメータリストの他の整数型パラメータを含む式である
- size_is は、ポインタ型の指す配列の大きさを示すもので、記憶域を用意する側の関数により配列の大きさが決定されることを基本とする。

in, inout, out, send パラメータにおいては、呼び元の関数により決定される。つまり、size_is の引き数には、in または inout パラメータを含めることができる。この場合、in 側の値のみ用いる。

receive においては呼び先の関数である。つまり、size_is の引き数には、in, out または inout パラメータを含めることができる。inout の場合には out 側の値が用いられる。

receive においては、例外として、呼び元により配列の大きさを決定することができる。この場合 size_is の引き数には in のパラメータが含まれる。

size_is で指定された領域は確保されているものとみなされる。in の const の場合であっても、size_is で示される領域全体についてアクセス可能でなければならない(count_is で示された領域はアクセス可能であって、size_is が

アクセス不能となることは許されない。このような問題のある状況は、例えばページ境界付近に配置された場合に起こりうる)

- `count_is` は、ポインタ型のパラメータを修飾するもので、ポインタが配列を指すことを表すと同時に、配列の有効な大きさを示す
- `count_is` は、記憶域に値を設定する側の関数により決定されることを基本とする。

`in`, `send` においては呼び元である (`count_is` の引き数には、`in` または `inout` パラメータを含めることができる)。

`out`, `receive` においては呼び先である (`count_is` の引き数には、`out` または `inout` パラメータを含めることができる)。

`inout` においては、`in` 方向では呼び元、`out` 方向では呼び先である (`count_is` の引き数には、`inout` パラメータを含めることができる)。

- `count_is` は省略することができる。

`count_is` が省略された場合、`size_is` の値に一致するとみなされる。

`size_is` の引き数が `in` 方向と `out` 方向で異なる値を持つ場合、`in` 方向の引き数を渡す場合には `in` 方向の値、`out` 方向の引き数を渡す場合には `out` 方向の値に一致するとみなされる。

【補足説明】以下の場合 `readv` において、`OpaqueRPC` では `buf` が不必要にコピーされる。

```
struct iovec {
    [size_is(size)]
    int8_t    *buf;
    int32_t    size;
};

signature sReadWriteSR{
    ER writev( [send(sPPAlloc),size_is(len)]struct iovec *io, [in]int32_t len);
    ER readv( [receive(sPPAlloc),size_is(*len)]struct iovec *io, [out]int32_t *len);
};
```

以下のように、`count_is`, `size_is` を分けて指定し、呼び元では `count` を 0 に指定することで、無用なコピーをさけることができる。

```
struct iovec {
    [size_is(size),count_is(count)]
    int8_t    *buf;
};
```

```

    int32_t    size;    // writev の場合、呼び先では書きこ
めた数を設定する
    int32_t    count;   // readv の場合、呼び元は0とし、呼
び先で読み出した数を設定する
                                // writev の場合、呼び先では0を設
定する。呼び元は通常 size と同じ値を設定
};

signature sReadWriteV {
    ER writev( [in,size_is(len)]struct iovec *st, [in]int32_t len );
    ER readv( [inout,size_is(len)]struct iovec *st, [in]int32_t len );
};

```

4.13.12 string 指定子の特性

- string は、ポインタ型のパラメータを修飾するもので、ポインタが NULL 終端された、または最大長さの指定された可変長の配列であることを示す

【補足説明】配列要素の型については、限定しない。NULL 終端は、整数型、浮動小数型、列挙型については、すべての bit が 0 のものを終端として扱う。struct 型についても同様にすべての bit が 0 のものを終端とする。

- string は、パラメータから最も遠いポインタ型を修飾する
- string と size_is(*1) を同時に指定することができる。これは、文字列配列を表す。この場合、パラメータは二重ポインタ型、または基本指定子が receive の場合には三重ポインタ型である
- string は引数を取ることができるが、引数は定数式、または同じパラメータリストの他の整数型パラメータを含む式である。string が引数をとる場合、最大長さが引数の式の値であることを表す
- 配列が最大長さの場合、NULL されない(最大長さで NULL が末尾にある場合を除く)

4.13.13 typedef された型について

- typedef された型の場合、元の型に置換した後の型に対して size_is(*1), string が適用される

- typedef で string 指定された型に対し、パラメータで string 指定することはできない (string のコンフリクト)。ただし、string の引数有無、引数がある場合に引数が一致する場合には、ジェネレータは警告を発生にとどめてもよい

*1 size_is の代わりに count_is のみ指定されていても同様

4.13.14 nullable 指定子

nullable 指定子は、引数の型がポインタ型である場合に指定できる。ただし nullable と size_is を同時に指定することはできない。

【補足説明】size_is において大きさ 0 と指定された場合、ポインタ値は NULL でなくてはならない。

多重ポインタの場合、最も右側 (変数に近い) のポインタが NULL となりうることを表す。ただし、基本指定子が receive の場合、右から 2 番目のポインタが NULL となりうることを表す。

【補足説明】どのポインタに対する指定子となるかは、size_is, count_is と同じである

nullable が指定されていないポインタには、必ず有効な記憶領域を指すポインタ値を与えなくてはならない。

nullable を指定した効果は、通常 TECS ジェネレータの出力には現れない。nullable が指定された引数を扱う場合、セルタイプコードにおいて NULL 値が渡されていないかを確認し、NULL でない場合のみポインタの指す記憶領域を参照してもよい。

receive 引数の場合、アロケートに失敗する可能性があるのであれば nullable を指定する必要がある

4.14 式

式は、本節で示す演算からなる。式は、コンポーネント記述が解釈されるときに評価される。

式を評価するのは、初期化子として式が与えられた場合と、size_is, count_is, string の引数として式が与えられた場合である。

関数引き数や構造体メンバの size_is, count_is, string で他のパラメータを参照している場合、定数として値が求まらないものについては、型の導出のみを行う。

【制限】整数については、無限精度により評価される。このため無符号整数と有符号整数との演算は、有符号で行われる。

【制限】型の格上げは暗黙的に行われるが、格下げは暗黙的に行われない。キャストが必要である。

【制限】文字列リテラルは、(char_t *) 以外の型にキャストできない

【制限】ブール型は、整数型にキャストしない限り、他の型との演算はできない

【制限】各演算子の意味については、説明を省略する。上記の制限を除き C 言語の演算子と同じ意味を持つ。

4.14.1 基本式

```
primary_expression
: IDENTIFIER
| 'true'
| 'false'
| INTEGER_CONSTANT
| FLOATING_CONSTANT
| OCTAL_CONSTANT
| HEX_CONSTANT
| CHARACTER_LITERAL
| string_literal_list
| '(' expression ')'
```

基本式における IDENTIFIER は、以下のいずれかである。

- 定数定義文により定義された定数
- 式が関数仮引き数の size_is, count_is, string 指定子の引き数として用いられる場合、参照される他の仮引き数
- 式が属性の size_is 指定子の引数として用いられる場合、参照される他の属性
- 式が構造体メンバ変数の size_is, count_is, string 指定子の引数として用いられる場合、参照される他の構造体メンバ変数
- 内部変数の初期化子

定数式の一部に C_EXP を用いることはできない。

【補足説明】C_EXP は単一の初期化子として用いるものであり、式の一部として用いることはできない。

4.14.2 文字列リテラルリスト

```
string_literal_list
: STRING_LITERAL
| string_literal_list STRING_LITERAL
```

単一の文字列、または、文字列の接続は文字列リテラルリストである。文字列リテラルリストに含まれる文字列は連結されて、一つの文字列として扱われる。

4.14.3 後置式

```
postfix_expression
: primary_expression
| postfix_expression '[' expression ']'
| postfix_expression '.' IDENTIFIER
| postfix_expression '->' IDENTIFIER
```

【制限】 関数呼び出しと後置インクリメント、デクリメント演算子がない

4.14.4 単項式

```
unary_expression
: postfix_expression
| unary_operator cast_expression
| 'sizeof' unary_expression
| 'sizeof' '(' type_name ')'
```

【制限】 前置インクリメント、デクリメント演算子がない

4.14.5 単項演算子

```
unary_operator
: '&'
| '*'
| '+'
| '-'
| '~'
| '!'
```

4.14.6 キャスト式

```
cast_expression
: unary_expression
| '(' type_name ')' cast_expression
```

4.14.7 乗除式

```
multiplicative_expression
: cast_expression
| multiplicative_expression '*' cast_expression
| multiplicative_expression '/' cast_expression
| multiplicative_expression '%' cast_expression
```

4.14.8 加減式

```
additive_expression
: multiplicative_expression
| additive_expression '+' multiplicative_expression
| additive_expression '-' multiplicative_expression
```

4.14.9 シフト式

```
shift_expression
: additive_expression
| shift_expression '<<' additive_expression
| shift_expression '>>' additive_expression
```

4.14.10 関係式

```
relational_expression
: shift_expression
| relational_expression '<' shift_expression
| relational_expression '>' shift_expression
| relational_expression '<=' shift_expression
| relational_expression '>=' shift_expression
```

4.14.11 等価式

```
equality_expression
: relational_expression
| equality_expression '==' relational_expression
| equality_expression '!=' relational_expression
```

4.14.12 and 式

```
and_expression
: equality_expression
| and_expression '&' equality_expression
```

4.14.13 exor 式

```
exclusive_or_expression
: and_expression
| exclusive_or_expression '^' and_expression
```

4.14.14 or 式

```
inclusive_or_expression
: exclusive_or_expression
| inclusive_or_expression '|' exclusive_or_expression
```

4.14.15 論理 AND 式

```
logical_and_expression
: inclusive_or_expression
| logical_and_expression '&&' inclusive_or_expression
```

4.14.16 論理 OR 式

```
logical_or_expression
: logical_and_expression
| logical_or_expression '||' logical_and_expression
```

4.14.17 条件式

```
conditional_expression
: logical_or_expression
| logical_or_expression '?' expression ':' conditional_expression
```

4.14.18 式

```
expression
    : conditional_expression
```

【制限】コンマ演算子が使えない

4.14.19 定数式

```
constant_expression
    : conditional_expression
```

4.15 型定義

型定義は、組み込み型を組合わせて新しい型を定義しなおすものである。C 言語における型定義と同様であり、同様に扱われる。

4.15.1 型定義文

```
typedef
    : 'typedef' type_specifier_qualifier_list declarator_list ';'
    | 'typedef' '[' typedef_specifier ']'
      type_specifier_qualifier_list declarator_list ';'
    ;
```

型定義文は `typedef` キーワードに続く型指定子修飾子リスト、宣言子リストからなる。末尾に区切り文字として `;` を置く。

`typedef` キーワードの後ろに `['', '']` で囲んだ型定義指定子を置くことができる。宣言子に含まれる識別子を型名として、型定義指定子修飾子リストおよび宣言子によって表される型が定義される。

【補足説明】型定義指定子は1つだけ置くことができる。

4.15.2 型定義指定子

```
typedef_specifier
    : 'string'
    | 'string' '(' expression ')'
```

型定義指定子として `string` キーワードまたは `string` キーワードに `'(', ')'` で囲まれた式を置くことができる。式は、定数値または定数からなる定数式であって、ジェネレータによる評価の結果定数値となる必要がある。

string キーワードは、ポインタ型を修飾する。ポインタ型以外は修飾できない。多重ポインタ型の場合、最も外側（左側）のポインタを修飾する。ポインタ型の指し示す型の配列であり NULL で終端されているか、式が指定されている場合には、最大でも式で与えられる長さの配列であることを示す。最大長さの配列の場合 NULL 終端されているとは限らない。

【参照実装における制限】型定義指定子は実装されていません。

4.15.3 組込み型と typedef 型

TECS CDL の言語仕様として定義する型には、TECS CDL の文法に予め組み込まれた組込み型と、typedef による型定義によって後から付け加えて使用する typedef 型とがある。

組込み型

値の範囲が明瞭な型

intN_t, uintN_t ... N: 8, 16, 32, 64, 128

float32_t, double64_t, bool_t, schar_t, uchar_t

char_t ... 入力値: -128 ~ 255 (有符号か無符号か不定)

値の範囲が不明瞭な型

C 言語由来の型 (char, int, short, long, double, float)

... 非推奨の型

typedef 型

t_stddef.h で定義される型

int_t, uint_t, long_t, ulong_t

intptr_t

tecs.h で定義される型

short_t, ushort_t

4.15.4 組込み型

組込み型は、TECS CDL の文法の一部として予め組み込まれた型である。C 言語に由来しない型でも TECS CDL 記述内では typedef することなく使用できる。C 言語に由来する型とは、char, short, int, long, float, double である (char, short, int, long は signed, unsigned で修飾することができる)。TECS CDL で、すべての C 言語由来の型が扱えるわけではない。short int, long signed, unsigned など、C 言語では有効な型であっても、TECS CDL 有効な型とは限らない (TECS CDL の文法定義参照)。

組込み型について import_C により取り込まれるヘッダファイルの中に組込み型 (C 言語に由来するものを除く) を定義する typedef が含まれていてもよい。少なくとも、TECS ジェネレータにより生成された C 言語のプロ

グラムをコンパイルする段階では、C 言語に由来しない組み込み型を定義する typedef 文が必要となる (TECS ジェネレータは TECS CDL の組み込み型を定義する typedef 文のコードを生成しない)。

intN_t が組み込み型である理由は、最大値、最小値が明瞭になり、値の範囲をチェックできるためである。intN_t を int や short などの型の typedef 型としてしまうと、最大値、最小値を tecsgen の段階では確定することができず、チェックできなくなってしまう。

4.15.5 typedef 型

ここで列挙した typedef 型は、TECS 仕様の一部とみなされる型である。typedef 型は typedef により定義される型であり、ヘッダファイルを取り込んで typedef による型定義をしない限り使用することができない。

4.15.6 tecs.h と t_stddef.h

組み込み型の C 言語での定義および、typedef 型を TECS CDL による記述で使用するために必要となる型の定義は tecs.h に含まれる。tecs.h は t_stddef.h を include する。t_stddef.h は TOPPERS 新世代カーネル統合仕様に基づいて定義されるものであり、TECS で定義する型以外の型定義を含む。

tecs.h では t_stddef.h で定義する型に加えて char_t, uchar_t, schar_t, short_t, ushort_t, int128_t, uint128_t が定義される。

通常 TECS CDL による記述では tecs.h を直接的または間接的に import_C で取り込む必要がある。ここで間接的にとは、import_C で取り込まれるヘッダファイルから include されることである。

【補足説明】typedef 型を TECS CDL による記述で用いないのであれば、必ずしも tecs.h を import_C で取り込む必要はないが、生成された C 言語のプログラムをコンパイルする際には、何らかの手段により tecs.h を include する必要がある。ターゲットが TOPPERS 新世代カーネル統合仕様に準拠しない環境である場合 tecs.h や t_stddef.h 以外で C 言語に由来しない組み込み型や typedef 型の定義をし、それらを import_C により取り込んでもよい。

4.16 ネームスペース

ネームスペースは、一般名前有効範囲を区切り、セルタイプ、シグニチャの名前衝突を防ぐために用いられる。

【補足説明】セルタイプとシグニチャのみに用いる。セルをリージョンに置くことで、ネームスペースと同じ効果が得られる。

【補足説明】以下のものも、ネームスペース下に置くことができない。

- 構造体 (struct) タグ
- 型定義 (typedef) された型名
- 定数 (const)
- 列挙型 (enum)

これらを TECS CDL でネームスペース下に置くことに技術的困難はないが、C 言語の生成コードにおいて短縮して記述する方法がなく、グローバル名を使用することになり利便性がないため。

4.16.1 ネームスペース文

```
namespace
    : 'namespace' namespace_name '{' statement_list '}' ';' ;
```

ネームスペースは `namespace` キーワードに、ネームスペース名に続き `;` で文リストを囲むことにより指定する。文リストにはネームスペース文を置くことができる。つまり再帰的にネームスペースを定義することができる。ネームスペースは、一般名前有効範囲に属する物の名前衝突を防ぐ。

4.16.2 ネームスペース名

```
namespace_name
    : IDENTIFIER
```

ネームスペース名は、単一の識別子である。

4.16.3 ネームスペース識別子

```
namespace_identifier
    : IDENTIFIER
    | '::' IDENTIFIER
    | namespace_identifier '::' IDENTIFIER
```

ネームスペース識別子は、一般名前有効範囲に属する物を参照するとき使用する。ネームスペース識別子には、以下の3種類がある。

- 単一の識別子 ... 同じネームスペースに存在する物を参照
- 絶対パス指定 ... ルートネームスペースから指定して、物を参照。
- 相対パス指定 ... 入れ子の内側のネームスペースに存在する物を参照

単一の識別子、および相対パス指定の先頭の識別子と一致するものがネームスペース内にないとき、上位のネームスペースから探索する。

一般名前有効範囲の項も参照。

4.17 シグニチャ

シグニチャは、関数ヘッダの集合であり、提供する、または利用する機能を示す。

関数は機能を提供するものであるが、数学的な関数と異なり、実際のプログラムでは一連の関数によって機能が達成される場合が多い。シグニチャは、ある目的機能を実現するために必要となる一連の関数を、一つの括りとして扱うためのものである。あるいは、独立しているが関連性の高い関数を一つの括りとして扱うためのものである。

シグニチャは、呼び口および受口に対応付けられて、それらの整合性を示す指標となる。その意味で、シグニチャは TECS において非常に重要である。しかし、シグニチャは構文的な整合性は保証するが、意味的な整合性を保証しない。意味的な整合性の保証について TECS 仕様では規定しない。この点は、各シグニチャの仕様書により規定する必要がある。

4.17.1 シグニチャ文

```
signature
    : 'signature' signature_name '{' function_head_list '}' ';' ;
```

シグニチャ文は、シグニチャを定義するためのものである。

シグニチャは、キーワード 'signature' に続いてシグニチャ名、'(', ') ' で囲まれた関数ヘッダリストからなる。

4.17.2 シグニチャ名

```
signature_name
    : IDENTIFIER
```

シグニチャ名は、識別子である。

4.17.3 関数ヘッダリスト

```
function_head_list
    : function_head
    | function_head_list function_head
```

関数ヘッダリストは、関数ヘッダのリストである。

4.17.4 関数ヘッダ

```
function_head
:
    type_specifier_qualifier_list declarator ';'
| '[' 'oneway' ']' type_specifier_qualifier_list declarator ';
```

関数ヘッダは、型指定子修飾子リストと宣言子からなる。宣言子としては、関数型のもののみ許される。つまり宣言子は、識別子に続き '('、')' で囲まれる引数リストを伴う形式である必要がある。

指定子として '['、']' で囲んだ `oneway` を指定することができる。

4.17.5 oneway 指定子

`oneway` 指定子は、呼び先から呼び元へ、一切の値を返すことではないことの指定である。一切の値を返さないとは `inout`, `out` または `receive` 指定された引数がないこと、戻り値を返さないことを意味する。

`oneway` が呼び先から呼び元へ、一切の値を返すことではないことの指定としているのはリモート呼び出し (RPC) を想定した仕様であり、RPC の場合には非同期呼び出しとなる。非同期呼び出しの場合、呼び先関数の実行要求の発行が終わった時点で、呼び元の関数は処理を再開する。このため呼び元関数と呼び先関数の並列処理が可能になる。

あるいは、複数の要求をまとめて転送する実装も考えられる。

一方 `oneway` 指定された関数で、RPC による呼び出しでなく、直接呼び出しの場合には、同期呼び出しとなる。

【仕様決定の理由】`oneway` の代わりに `async` を指定子とする案が、仕様策定の当初あった。しかし、シグニチャの関数が必ずしも RPC で呼び出されるとは限らず、直接呼び出しとなる場合も想定される。直接呼び出しの場合には、必ず同期呼び出しとなるため `async` を避けた。

RPC プラグインでは、デフォルト動作としては一方向関数は非同期呼び出しとして、プラグインのオプションで `noasync` が指定された場合に一方向関数であっても同期呼び出しとする。

`oneway` 指定された関数は、ER 型または void 型を返すものでなくてはならない。ER 型を返す場合、呼び出された側では E_OK 以外の値を返してはならない。先に述べたとおり `oneway` では戻り値を返すことはできない。その意味では、関数の戻り値の型は void 型の関数であるべきである。しかし、RPC 実装となった場合には、マーシャラレベルのエラー（通信チャンネルで発生するエラー）を返す可能性がある。このため RPC レベルでエラーを返すことを想定して ER 型も可としている。RPC 実装の場合には、呼ばれ

た側がどんな値を返しても呼び元に返されることはなく、マーシャラレベルでのエラーがなければ E_OK が返される。一方、直接呼び出しの実装の場合には、呼ばれた側の ER 型の値が返されてしまうが、直接呼び出しではマーシャラレベルのエラーが発生することはありません、必ず E_OK が返されなくてはならない。

【補足説明】現在の TECS 仕様では ITRON 仕様への適合性を考慮して、エラーを返すのは ER 型であると仮定されている。ITRON 仕様に適合しない実装では、エラーを返す型が ER 型とは限らないが、その場合のエラーを渡す型へ適合する方法が考慮されていない。同様に正常を示す値が E_OK であると仮定されている。

4.17.6 アロケータシグニチャ

アロケータは、メモリを割付けるコンポーネントのことである。アロケータシグニチャは、アロケータの受け口が持つシグニチャのことである。

アロケータシグニチャは、send または receive の引き数として与える。

アロケータのシグニチャは、少なくとも allocate と deallocate 関数を持たなくてはならない。

alloc 関数の第一引数は、アロケートしようとするメモリ領域のサイズ(バイト数)を表すものでなくてはならない。alloc 関数の第二引数は、アロケートされたメモリ領域へのポインタを返すポインタでなければならない(二重ポインタ)

dealloc 関数の第一引数は、デアロケートしようとするメモリ領域へのポインタでなくてはならない。

alloc, dealloc が、2 つ以上の引数を持つとき、through プラグインによって RPC チャネルを生成させるのに、関数の引数に与えるべき値をプラグイン引数として指定する。仮引数名によってどの引数に対する値かを識別する。このため alloc と dealloc で同じ仮引数名が指定されると、これらは同じ値を指定されることになる。もし、異なる値を指定する必要があるのであれば、シグニチャの設計者は、これらに異なる仮引数名を与える必要がある。関数の引数に与えるべき値として、定数または他の引数を指定できる。

alloc, dealloc 関数に引き数を追加してもよい。また、アロケータシグニチャに realloc 関数などを追加してもよい。

アロケータシグニチャの事例を以下に示す。

```
signature sAlloc {
    ER alloc( [in]size_t len, [out]void *p );
    ER dealloc( [in]void *p );
};
```

4.18 セルタイプ

セルタイプは、セルの特性を示すためのもので、どのような呼び口、受け口や属性を持つかなどを指定する。セルタイプは、コンポーネントすなわちソフトウェア部品の型式の位置づけである。

4.18.1 セルタイプ文

```
celltype
    : 'celltype' celltype_name '{' celltype_statement_list '}' ';' ;
```

セルタイプ文は、"celltype" キーワードに続いて、セルタイプ名と "{", "}" で囲まれたセルタイプ文リスト、末尾に ";" を置く。

4.18.2 セルタイプ名

```
celltype_name
    : IDENTIFIER
```

セルタイプ名は単一の宣言子である。

4.18.3 セルタイプ文リスト

```
celltype_statement_list
    : specified_celltype_statement
    | celltype_statement_list specified_celltype_statement
```

セルタイプ文リストは、指定子セルタイプ文のリストである。

4.18.4 指定子セルタイプ文

```
specified_celltype_statement
    : celltype_statement
    | '[' celltype_statement_specifier ']' celltype_statement
```

指定子セルタイプ文は、セルタイプ文、または '[' , ']' に囲まれたセルタイプ文指定子に続いてセルタイプ文を記述する。

4.18.5 セルタイプ文指定子

```
celltype_statement_specifier
: 'inline'
| 'allocator' '(' alloc_list2 ')'
| 'optional'
```

セルタイプ文指定子は 'inline'、'allocator' または 'optional' である。

指定子 'inline' と 'allocator' とは、受け口に対してのみ指定できる。それ以外のセルタイプ文に対する指定は誤りである。

'inline' が指定された受け口は、受け口関数が inline 関数として実装されることを表す。実装については、次章にて規定する。

'allocator' 指定子は引数としてアロケータリスト 2 を持つ。

'optional' 指定子は、呼び口にのみ指定できる。呼び口を未結合のままとしてもよいことを表す。未結合の呼び口を呼び出してはならない。セルタイプコードにおいて、結合されているかどうかをテストした上で、結合を確認した後に呼び出さなくてはならない。

4.18.6 アロケータリスト 2

```
alloc_list2
: alloc2
| alloc_list2 ',' alloc2
```

アロケータリスト 2 は、アロケータ指定子 2、あるいはアロケータ指定子 2 を ',' でつないだリストである。

```
alloc2
: IDENTIFIER '.' IDENTIFIER '=' initializer # (1)
| IDENTIFIER '.' IDENTIFIER '<=' initializer # (2)
```

アロケータ指定子 2 には、2 種類がある。デバイスドライバ向けのセルフアロケータ (1) と多段リレーモデル向きのリレーアロケータ (2) 指定がある。

【参照実装における制限】セルフアロケータは未実装。

リレーアロケータの場合、'i=' の左辺に関数名とパラメータ名を '.' でつないだものである。パラメータは send または receive でなければならない。

右辺は、呼び口名、関数名、パラメータ名を '.' でつないだものである。この呼び口は、同じセルタイプの範囲で先方参照が可能である。パラメータは send または receive でなければならない。

リレーアロケータを指定するのは、二通りある。

- アロケートされたものを呼び元から、そのまま呼び先に受け渡す場合

- 反対に呼び先から受け取ったものに呼び元に渡す場合

【未決定事項】右辺に受け口を指定できない。できてもよいかもしれない。

【参照実装における制限】受け口配列のリレーアロケータは未実装。

4.18.7 セルタイプ文

```
celltype_statement
  : port
  | attribute
  | var
  | require
  | factory
```

セルタイプ文は、以下のいずれかである。

- 口文
- 属性文
- 内部変数文
- リクワイア文
- ファクトリ文

これらのうち、口文と属性文を合わせて「インタフェース表明」と呼ぶ。残りの部分を「セルタイプの実装」または単に「実装」と呼ぶ。

4.19 呼び口、受け口

呼び口、受け口は、セルを他のセルと結合する時の窓口となる。呼び口は機能を要求する側の口であり、受け口は機能を提供する側の口である。呼び口、受け口は、一つのシグニチャに対応付けられる。呼び口、受け口は、TECS CDL においては口文として定義される。

4.19.1 口文

```
port
  : port_type namespace_signature_name port_name ';'
  | port_type namespace_signature_name port_name
    '[' ']' ' ';
  | port_type namespace_signature_name port_name
```

```

    '[' array_size ']' ';'
  | port_type namespace_signature_name port_name
    '<' namespace_identifier '.' IDENTIFIER ';'

```

口文は、口タイプ、ネームスペースシグニチャ名、口名、そして末尾の ';' からなる。末尾に "[]" または "[array_size]" を付加して、配列であることを示すことができる。呼び口の場合には呼び口配列、受け口の場合は受け口配列と呼ばれる。

配列の場合、配列サイズを指定することも、省略することもできる。配列サイズが省略された場合、配列の大きさは組上げ記述において決定される。

最後の形式は、固定結合を指定するものである。

4.19.2 口タイプ

```

port_type
  : 'call'
  | 'entry'

```

口タイプは、以下のいずれかである。

- 呼び口 'call'
- 受け口 'entry'

4.19.3 ネームスペースシグニチャ名

```

namespace_signature_name
  : namespace_identifier

```

ネームスペースシグニチャ名は、ネームスペース識別子である。ネームスペース識別子の指し示すものは、シグニチャでなければならない。

4.19.4 口名

```

port_name
  : IDENTIFIER

```

口名は、単一の識別子である。

4.19.5 配列サイズ

```
array_size
    : constant_expression
```

配列サイズは、定数式である。定数式は、評価の結果、正の整数値を持つ必要がある。

4.19.6 固定結合

固定結合は、セルタイプの受け口の定義において、呼び元のセルを指定するものである。

【補足説明】初期化のための結合を、一々記述することなく、省略することを目的とした機能である。

固定結合は、以下の条件を満たす必要がある。

- 固定結合先の呼び口は、添数なしの呼び口配列でなくてはならない
- 固定結合先にセルタイプを指定する場合、そのセルタイプは、シングルトンでなくてはならない
- 固定結合先にセルを指定する場合、そのセルタイプはシングルトンでなくてもよい
- 固定結合指定されている受け口は、非配列でなくてはならない
- 複合セルタイプの内部セルが固定結合が指定されている受け口を持つ場合、複合セルタイプの外部のセルから直接結合される

複合セルタイプの内部セルが固定結合が指定されている受け口を持つ場合、複合セルタイプの外部のセルから直接結合されるため、複合セルタイプにおいて、外部結合のための受け口を持つことはない。

固定結合の結合順序は、以下の通りである。

- ソースプログラム中のセルの出現順（構文解釈が行われた順）に、固定結合の結合がなされる
- 複合セルタイプの内部セルにおいて、固定結合がなされている場合、複合セルタイプのセルの出現順番で結合がなされる
- 複合セルタイプの内部セルに複数の固定結合が指定されたセルがある場合、複合セルタイプ定義で内部セルが出現した順番で結合がなされる
- スループラグインで挿入されるセルは、すべてのソースプログラムの構文解釈が終わった後に構文解釈されるため、固定結合の順序としては後ろの方に置かれる

【補足説明】セルの id は、同じセルタイプに属するセルの間で順序付けするものである。固定結合では、複数のセルタイプが関わることを想定しており、出現順序で

4.20 属性

属性は、セルの持つ値を保持するものである。属性には、初期値を与えることができるが、実行時に書き換えることはできない、定数として扱われる。属性の初期値は、セルタイプの定義時だけでなく、セルを定義する際にも与えることもできる。両方に初期値が与えられた場合には、セルにおける初期値が優先される。属性の初期値は、セルタイプかセルの少なくとも一方の定義において、与えられる必要がある。

4.20.1 属性文

```
attribute
    : 'attr' '{ attribute_declaration_list }' ';' ;
```

属性文は、キーワード 'attribute' に ', ' で囲まれた属性宣言リストと、末尾の ';' からなる。

4.20.2 属性宣言リスト

```
attribute_declaration_list
    : attribute_declaration
    | attribute_declaration_list attribute_declaration
```

属性宣言リストは、属性宣言のリストである。

4.20.3 属性宣言

```
attribute_declaration
    : declaration
    | '[' attribute_specifier ']' declaration
```

属性宣言は、宣言文である。この宣言文は、変数または配列でなくてはならない。属性は、セルが記憶する値である。

宣言文では、初期化子により属性の初期値を指定できる。初期化子に現れる定数式には、定数定義文で定義された定数の識別子を含めることができる。属性の初期値は、セルタイプまたはセルの少なくとも一方において与えられ

なければならない。属性の初期値は、セルの定義において初期値の指定があった場合には、そのセルにおいてはその初期値が優先される。

変数がポインタ型の場合、ポインタの指すものが配列の場合 `size_is` 指定子により配列サイズを指定する。この場合、ポインタが指す配列が `size_is` 指定子により指定された大きさとなるように、ジェネレータが調整する。つまり、初期化子の要素数が少ない場合には、不足する分、追加される。

4.20.4 属性指定子

```
attribute_specifier
  | 'omit'
  | 'size_is'
```

属性指定子は、以下のいずれかである。

- cell CB への出力をしない 'omit'
- 配列サイズ 'size_is'

cell CB への出力をしない 'omit' は、その属性が、ファクトリ文で利用する目的であって、cell CB への出力する必要がない場合に指定する。

配列サイズ 'size_is' は、その属性がポインタ型のときのみ指定できる。ポインタ型が配列を指している、そのサイズが `size_is` の引数で示される大きさを持つことを指定できる。

`size_is` は、式を引数にとる。式は定数式、または属性を含む式である。この属性は、前方参照が許される。

【補足説明】 `size_is` の引数で示された大きさを持つ配列であることが指定できる。配列の初期化子の要素数に応じて `size_is` の引数として指定された属性を初期化する手段としては使えない。

4.21 内部変数

内部変数も、セルの持つ値を保持するものであるが、属性とはいくつかの点で異なる。

- 実行時に書き換えることができる
- セルの定義時に初期値を与えることはできない
- 初期値を指定するのに属性を参照することができる
- 初期値をまったく与えないままとすることができる

内部変数は、セルの定義において直接的に初期値を与えることはできないが、属性を参照することで間接的に初期値を与えることができる。

4.21.1 内部変数文

```
var
    : 'var' '{' var_declaration_list '}' ';' ;
```

内部変数は、キーワード `var` に `;` で囲まれた内部変数宣言リスト、末尾の `;` からなる。

4.21.2 内部変数宣言リスト

```
var_declaration_list
    : var_declaration
    | var_declaration_list var_declaration
```

内部変数宣言リストは、内部変数宣言のリストである。

4.21.3 内部変数宣言

```
var_declaration
    : declaration
    | '[' var_specifier ']' declaration
```

内部変数宣言は、宣言文、または `[`, `]` で囲まれた内部変数指定子に続く宣言文である。

宣言文の初期化子の式には、定数式その他、属性を含む式を用いることができる。

ただし、属性を含む式は、整数型、浮動小数型、文字型、ブール型、ポインタ型の初期化子に限られる。構造体型の初期化子には用いることができない。

内部変数に初期値が省略された場合、初期値として、整数型や浮動小数型の場合には 0、ブール型の場合には false が与えられたものとして扱われる。また、ポインタ型は size_is 指定子が指定されている場合には、size_is 指定子の引き数で示される大きさの配列が生成されて割付けられる。それ以外のポインタ型の場合、0 が初期値として与えられたものとして扱われる。

内部変数は、セルにおいて初期値を与えることはできない。

【補足説明】内部変数は、セルにおいて初期値を与えることはできないが、セルタイプにおいて内部変数の初期値に属性を与えることで、セルにおいて間接的に初期値を与えることができる。ROM に置かれた属性から RAM 上の内部変数を初期化することを想定した仕様であり、内部変数を RAM 上の未初期化領域に置くことで、初期値を記憶するための ROM 領域の消費を抑えるものである。

【参照実装における制限】参照実装においては、`;` の内側の初期化子には、上記可能な型においても属性を用いることができない。

4.21.4 内部変数指定子

```
var_specifier  
    : 'size_is' '(' expression ')'
```

属性指定子は、以下である。

- 配列サイズ 'size_is'

配列サイズ 'size_is' は、その属性がポインタ型のときのみ指定できる。ポインタ型が配列を指している、そのサイズが size_is の引数で指定される大きさを持つことを指定できる。

size_is の引き数、size_is を指定されたポインタ型の変数、およびポインタの指す配列は、以下の特性を持つ。

- size_is は式を引数にとる。式は定数式、または属性を含む式である。
- ポインタ型の内部変数は配列を指す定数として扱われ、書換えることはできない。
- ポインタの指す先の配列の要素は書き換えることができる。
- size_is の引数で指定された大きさの領域が確保される

(初期値の指定の有無に関係なく、size_is の引数で指定された大きさの領域が確保される)

- 初期値として ", " で囲んだ定数式を要素とする初期化子リストを指定できる

(初期化子リストの個数は size_is の引数で与えられる配列の大きさ以下の個数が指定できる)

4.22 リクワイア

リクワイア文により、セルタイプに属するすべてのセルについて、シングルトンセルの受け口への結合を指定する。

リクワイア文により指定されるシングルトンセルの受け口への結合は、暗黙的でありコンポーネント図において明示しない。

4.22.1 リクワイア文

```
require
    : 'require' IDENTIFIER '.' IDENTIFIER ';'
    | 'require' IDENTIFIER '=' IDENTIFIER '.' IDENTIFIER ';'

```

リクワイア文は、キーワード ”require” に続きセルタイプ名またはセル名、さらに ’.’ に続き受け口名、そして末尾に ’;’ を置く。あるいは、キーワード ”require” に続き、’=’ の左辺にリクワイア呼び口名、右辺にセルタイプ名またはセル名、’.’ に続き受け口名、そして末尾に ’;’ を置く。

ここで、セルタイプ名を指定する場合、そのセルタイプはシングルトンでなくてはならない。受け口が、受け口配列であってはならない。

セル名を指定する場合、そのセルを前方参照できる。

リクワイア文は、リクワイア呼び口と、リクワイア呼び口の結合先の受け口を指定するものである。

リクワイア呼び口は、通常の呼び口とはいくつかの点で異なる。

- 呼び口名を与えないでおくこともできる
- セルタイプにおいて、結合を定義する
- 呼び先を指定するのに、セル名に代えてセルタイプ名で指定することもできる

リクワイア呼び口名は、他の口名、属性名、内部変数名と重複した名前とすることができない。リクワイア呼び口名が与えられたリクワイア呼び口への結合を、セルでオーバーライドすることがことができるが、処理系は警告を発する。

リクワイア文で指定された受け口は、このセルタイプに属するすべてのセルにおいて同じセルの同じ受け口に結合される。このため、各セルの定義においては、受け口への結合を指定しない。各セルの定義において、リクワイア文による結合は暗黙的である。

リクワイア文は、一つのセルタイプにおいて複数記述することができるが、それぞれの受け口のシングニチャに属する関数の名前が重複することはできない。実装においては、関数名のみを指定する。

リクワイア文では、受け口を指定するのにセル名、セルタイプ名のいずれかを指定できる。通常の呼び口の結合では、セル名しか指定できないが、リクワイア文では結合先がシングルトンセルに限定されるため、セルタイプ名が指定されても、セルが特定される。

あるシングルトンセルタイプのセルが複数存在する場合があります。別にコード生成されるリージョンごとにシングルトンセルタイプのセルが存在する場合である。この場合、セルタイプ名を指定すると、同時に生成されるシ

シングルトンセルタイプセルを参照する。同時に生成されるセルがない場合は、誤りである。

4.23 ファクトリ

ファクトリは、ITRON 仕様の OS においてはコンフィギュレーションファイルを作成する目的で使用する。ファクトリには、セルタイプのファクトリ、つまりセルタイプが指定されたことにより必要となる資源を生成する場合と、セルごとに必要となる資源を生成する場合がある。

なお、ファクトリにより生成する文字列の出力先は、コンフィギュレーションファイルに限定されない。

4.23.1 ファクトリ文

```
factory
    : factory_head '{' factory_function_list '}' ';' ;
```

ファクトリ文は、ファクトリ頭部と ”, ” で囲まれたファクトリ関数リスト、末尾の ”;” から成る。

4.23.2 ファクトリ頭部

```
factory_head
    : 'factory'
    | 'FACTORY'
```

ファクトリ頭部は、セルタイプのファクトリを示すキーワード ”FACTORY” とセルのファクトリを示すキーワード ”factory” のいずれかである。

セルタイプのファクトリは、そのセルタイプのセルが一つ以上生成される場合において評価が行われる。

セルのファクトリは、セルが一つ生成されるごとに評価が行われる。

4.23.3 ファクトリ関数リスト

```
factory_function_list
    :
    | factory_function_list factory_function
```

ファクトリ関数リストは、0 個以上のファクトリ関数のリストである。

4.23.4 ファクトリ関数

```
factory_function
  : factory_function_name
    '(' constant_expression ',' constant_expression ')' ';'
  | factory_function_name
    '(' constant_expression ',' constant_expression ',' arg_list ')' ';' ;
```

ファクトリ関数は、ファクトリ関数名と '(' ; ')' に囲まれる引数から成る。ファクトリ関数名として "write" を指定できる。

第一引数は、出力先のファイル名を指定する文字列である。コンフィギュレーションファイルの場合、"tecsген.cfg" を指定することを推奨する。

第二引数は、第一引数で指定されたファイルに出力するフォーマット文字列を指定する。この文字列は C の printf 文のフォーマット指定と同様に扱われる。例えば、フォーマット文に

第三引数以降は、オプションであり、フォーマット文字列によって引数の有無が決定される。

【参照実装における制限】 write 以外ファクトリ関数は定義されていない。

【参照実装における制限】 参照実装において、フォーマットはコード生成段階において評価される。このため多少ファイル出力が行われる。

【参照実装における制限】 整数値を出力する場合でも、C_EXP が指定された場合に備えて、フォーマットでは

4.23.5 ファクトリ関数名

```
factory_function_name
  : IDENTIFIER
```

ファクトリ関数名は識別子である。

4.23.6 ファクトリ引数リスト

```
arg_list
  : IDENTIFIER
  | arg_list ',' IDENTIFIER
```

引数リストは、識別子または識別子を ',' で連結したリストである。識別子は、以下の順にサーチされる。

- 属性 (attribute)
- 定数

識別子は、その右辺値に置き換えられて、write 関数の引数として渡される。属性の場合、右辺値は、セルに定義されていればその値を、セルに定義されていなければセルタイプで定義されたデフォルトの値を、それも定義されていなければ0または空文字列となる。

【補足説明】引数として式を与えることはできない。

4.23.7 名前置換

write 関数の引数の文字列定数に含まれる \$id\$ などの文字列は置換が行われる。以下に置換の行われる文字列の一覧を示す。

```
$id$      ... セルタイプ名とセル名を '_' で連結したものに置換
$cell$    ... セル名に置換
$cb$      ... セルの CB の C 言語名に置換
$cbp$     ... セルの CB へのポインタ (CB が生成されない場合は NULL に置換)
$cb_proto$ ... セルの CB の C 言語名 (プロトタイプ宣言用) に置換
$ct$      ... セルタイプ名に置換
$idx$     ... セルの CB の IDX (idx_is_id の場合は整数、そうでない場合は CB へのポインタ) に置換
$ID$      ... セルの ID(idx_is_id の場合 IDX に一致) に置換
$$        ... $ に置換
```

最後の規則が優先され、例えば \$\$id\$ は \$id\$ に、\$\$ct\$ は \$ct\$ に置換される。また、セルタイプファクトリ (FACTORY) の内側では \$ct\$, \$\$ の置換のみが行われる。

4.23.8 ファクトリヘッダ

ファクトリヘッダは、セルタイプコードによって取込まれることを意図したヘッダファイルである。セルタイプ tCelltype に対するセルタイプファクトリヘッダは、以下のファイルである。

```
tCelltype_factory.h
```

具体的には TOPPERS/ASP のコンフィグレータが生成する kernel_cfg.h を取込むことである

```
factory {
    write( "$ct$_factory.h", "#include \"kernel_cfg.h\" );
};
```

4.24 セル

セルは、TECS におけるコンポーネント (ソフトウェア部品) にあたり、セルタイプのインスタンスである。

セル文において、呼び口の結合先と、属性の初期値を与えて、セルを定義する。

4.24.1 セル文

```
cell
    : 'cell' namespace_celltype_name cell_name '{' join_list '}' ';'
    | 'cell' namespace_celltype_name cell_name ';' ;
```

セル文は、セルを定義する (静的生成させる) またはプロトタイプ宣言するためのものである。

セル文は、キーワード "cell" に続いてネームスペースセルタイプ名、セル名、", " で囲まれた結合リスト、そして末尾の ";" からなる。

4.24.2 ネームスペースセルタイプ名

```
namespace_celltype_name
    : namespace_identifier
```

ネームスペースセルタイプ名は、ネームスペース識別子である。

4.24.3 セル名

```
cell_name
    : IDENTIFIER
```

セル名は、識別子である。

4.24.4 プロトタイプ宣言

プロトタイプ宣言は、セルが定義される前に参照可能とするために、セルの名前と属するセルタイプまたは複合セルタイプを宣言するものである。あるいは、セルを定義する前に、セルの一部または全部の属性、結合先、非結合先を指定する目的で用いることができる。

プロトタイプ宣言には、純粹プロトタイプ宣言と指定プロトタイプ宣言の2通りがある。

4.24.5 純粋プロトタイプ宣言

純粋プロトタイプ宣言は、セルを前方参照可能とする目的で用いられる。

TECS CDL ではセルの前方参照が可能であり、通常純粋プロトタイプ宣言を用いる必要はないが、意味解析後に生成され解釈されるポストコードで生成されるセルは、意味解析段階で参照できないため、プロトタイプ宣言が必要となる

記述例を以下に示す。

```
cell tCelltype Cell;
```

”,” で囲まれた結合リストを伴わない。

4.24.6 指定プロトタイプ宣言

指定プロトタイプ宣言は、プロトタイプ宣言で、呼び口や属性の一部または全部のを予め定義しておく目的で用いられる。プロトタイプ宣言されたセルの定義が行われなければ、セルは生成されない。

セルの定義では、プロトタイプ宣言された結合を変更することはできない(二重定義エラーとなる)。プロトタイプ指定子とジェネレート指定子を同時に指定できない。セルの定義が行われたに、そのセルの指定プロトタイプ宣言を行うことはできない。

【補足説明】 ジェネレート指定子は構文解釈時にプラグインを呼び出す。意味定義段階でプラグインを呼び出すと、セル生成が不適切になる可能性がある。

prototype を指定された cell は、何度でも同名で定義できる

記述例を示す。

```
[prototype]          // 一部の結合だけを指定する
cell tMotorA MotorA {
    eHandler <= GPIO_16bit_0.cIRQ[3]; // 逆 requier(固定結合)
    cGPIO = GPIO_16bit_0.eGPIO[3];    // GPIO_16bit No.0 の bit 3 を
要求
};
```

4.24.7 セルの id 番号

文指定子の id 指定子によりセルの id 番号を指定することができる。id 指定子は、id 番号を引数にとる。id 番号は、正または負の整数である。

id = 0 や文字列などはエラー

id 番号はセルタイプごとに 1 から (生成されるセルの個数) までの番号を指定できる。

id 番号に範囲外となるものを指定できない

id 番号が重複することはできない

セルに id 指定子が指定されていない場合、出現順に id = 1 から順に与えられる

ただし id 指定されている番号は避けられる

TECS ジェネレータは id 番号順に、コードを生成する。セルの生成順序を TECS CDL の出現順序とは別に制御したい場合に用いる。

【補足説明】tecsgen のオプション -u を指定すると、全体の通し番号となるが、セルタイプごとには連続した値が与えられる。その場合でも id = 1 のセルがセルタイプにおいて最初のセルとなる。tecsgen のオプション -v を指定すると、最終的に与えられた id 番号が表示される。

id 指定の例を以下に示す。

```
[id(-1)]           // 一番後ろの ID となる (5)
cell tCelltype Cell15{
};

cell tCelltype Cell12{ // 未指定 id = 2 となる
};

cell tCelltype Cell13{ // 未指定 id = 3 となる (出現順)
};

[id(-2)]           // 後ろから 2 番目の ID となる (4)
cell tCelltype Cell14{
};

[id(1)]            // 一番先頭の ID となる (1)
cell tCelltype Cell11{
};
```

4.25 結合

TECS において結合とは、呼び口に受け口を割付けることである。

TECS CDL の構文においては、便宜上、セルにおいて属性の初期値を指定する場合も結合として扱う。

なお、内部変数は、セルにおいて初期化できず、結合として扱われない。

4.25.1 結合リスト

```
join_list
:
| join_list specified_join
| join_list reverse_join
```

結合リストは、空、または指定子リスト付き結合文のリスト、または逆結合である。

4.25.2 指定子リスト付き結合文

```
specified_join
: '[' join_specifier_list ']' join
| join
```

指定子付き結合は、結合である。'[', ']' に囲まれた結合指定子リストを前に置くことができる。

4.25.3 結合指定子リスト

```
join_specifier_list
: join_specifier_list ',' join_specifier
| join_specifier
```

結合指定子リストは結合指定子を ',' で結合したリストである。

4.25.4 結合指定子

```
join_specifier
: 'through' '(' plugin_name ',' plugin_arg ')'
```

結合指定子は、スルー指定子で、キーワード "through" と '(' , ')' で囲まれたプラグイン名、',' と、プラグイン引数からなる。

4.25.5 スルー指定子

スルー指定子は、呼び口にスループラグインの適用を指示する。

プラグイン名で指定されたプラグインがジェネレータにロードされ、ジェネレータがスルーセルを生成する。スルーセルとは、呼び口と右辺で指定された受け口の間に挿入されるセルのことである。

4.25.6 プラグイン名

```
plugin_name
    : IDENTIFIER
```

プラグイン名は、識別子である。

4.25.7 プラグイン引数

```
plugin_arg
    : string_literal_list
```

プラグイン引数は、文字列定数である。

プラグイン引数は、プラグインモジュールによって解釈されるため各プラグインの仕様に依存するが、以下の仕様を基本とする。

- ・ '=' の左辺にパラメータ名、右辺に文字列を置く
- ・ 左辺のパラメータ名は、識別子である
- ・ つまりパラメータ名は、先頭文字はアルファベットか '_' で、2文字目以降はアルファベット, '_' または数字である
- ・ ', ' で区切るにより、複数のパラメータをプラグイン引数として渡すことができる
 - ・ 右辺文字列の前後の空白文字は取り除かれる (\ " で囲まれている場合を除く)
 - ・ 右辺文字列中のダブルクォート '"' は、バックスラッシュ '\ ' でエスケープする必要がある
 - ・ 右辺文字列中にカンマ ', ' を含む場合には、右辺文字列全体を '\ "' で囲む必要がある
 - ・ 右辺文字列中にダブルクォート '"' とカンマ ', ' を含む場合には、右辺文字列全体を '\ "' で囲む必要がある
 - ・ さらに右辺文字列中のダブルクォート '"' にカンマ ', ' が続く場合は、カンマ ', ' もバックスラッシュ '\ ' でエスケープする必要がある

以下に例を示す。

```
"param0 = val str"           ... '=' の左辺にパラメータ名、
                             右辺に値の文字列
"param1 = val str, param2 = val str2" ... ', ' で連結
"param3 = \"val str, val str2\"" ... 右辺文字列が ', ' を含む場
                             合 \" で囲む
"param4 = C_EXP( \"MAIN_PRIORITY\" )" ... 右辺文字列が '"' を含
                             む場合
```


optional 指定されている呼び口の場合には、未結合の添数があってもよい。添数の出現順序は規定しないが、同じ添数が重複してはならない。

添数が指定されない場合、出現順に 0 から (出現個数 - 1) まだが添数として与えられる。セルタイプの呼び口の定義で配列サイズが指定されている場合、配列サイズ分の定義をしなくてはならない。

以下の例のように、一つの呼び口配列について、添数を指定する場合と、指定しない場合を混在することはできない。

例) 不適切な例

```
cCall[0] = Cell.eEnt;
cCall[] = Cell.eEnt;
```

【補足説明】配列の全体を結合するには、すべての添数について結合を記述する。添数なしの呼び口と受け口を指定して全体を結合する手段はない。

4.25.10 属性の初期化の場合 (複合コンポーネントの内部セルの場合を除く)

結合により、セルの属するセルタイプで定義されている属性について、初期値を与えることができる。結合により属性の初期値が与えられない場合、セルタイプにおいて属性に与えられた初期値により初期化される。いずれにおいても属性の初期値が与えられない場合は、誤りである。

属性の場合、左辺の結合名は、属性名である。右辺は、属性の型に合う適当な初期化子を置く。初期化子に現れる定数式には、定数定義文で定義された定数の識別子を含めることができる。

属性が整数型、浮動小数型、ブール型の場合、初期化子は定数式である。

属性がポインタ型で `size.is` 指定された配列の場合、初期化子は ”,” で囲んだ初期化子リストである。非配列の場合、初期化子はポインタ型にキャストされた定数式である。

属性が構造体型の場合、初期化子は ”,” で囲んだ初期化子リストである。

4.25.11 複合コンポーネントの内部セルの属性の場合

結合が、複合セルタイプの内部セルの内側に現れた場合は、内部セルの属性を初期化するか、内部セルの属性を複合コンポーネントの属性に割付けるものになる。

内部セルの属性を初期化する場合は、前節の属性の初期化の場合と同じである。

内部セルの属性を複合コンポーネントの属性に割付ける場合、結合の割付記号 '=' の右辺の記号 '.' の左辺に 'composite' を置くことができる。この場合、記号 '.' の右辺の識別子は、複合セルタイプ属性文で定義される複合セルタイプが外部に公開する属性の名前である。複合セルタイプの属性名が、内部セルの持つ属性、変数、呼び口、受け口のいずれの名前とも重複しない場合には、'compsote' '.' を前置きすることなく、割付け記号 '=' の右辺に複合セルタイプの属性名のみを置くことができる。

内部セルの属性を複合コンポーネントの属性に割付ける場合、両者の型が一致しなくてはならない。また、内部セルの属するセルタイプにおいて属性の初期値が指定されていたとしても、この初期値は参照されない。複合セルタイプの属性または、複合セルタイプから生成されるセル（複合セルタイプに属するセル）のいずれか、少なくとも一方において、初期値が指定されなくてはならない。両方で指定された場合、セルで指定された初期値が優先される。

4.25.12 逆結合

reverse_join

```
: cae_name '<=' namespace_identifier '.' IDENTIFIER ';'
| cae_name '<=' namespace_identifier '.' IDENTIFIER '[' expression ']' ';'
| cae_name '[' array_index ']' '<=' namespace_identifier '.' IDENTIFIER ';'
| cae_name '[' array_index ']' '<=' namespace_identifier '.' IDENTIFIER '['
```

逆結合は、受け口を持つセルにおいて、呼び口を指定するものであり、結合とは逆の記述である。プロトタイプ指定されたセルにおいてのみ記述できる。セルのプロトタイプ宣言のみがなされて、定義がなされていない場合、呼び元のセルに結合は追加されない。

【補足説明】呼び元において受け口の情報を持つことから、呼び元のセルから結合先を指定するのが一般的な使用方法であり、逆結合の使用は最小限にとどめるべきであることから、プロトタイプ指定されたセルにおいてのみ記述可能とした。

cae_name は受け口名である。

以下の例では、呼び元のセル GPIO_16bit_0 に cIRQ[3] = MotorA.eHandler; が追加される。

```
[prototype] // プロトタイプ宣言指定子
cell tMotorA MotorA {
    eHandler <= GPIO_16bit_0.cIRQ[3]; // 逆結合
    cGPIO = GPIO_16bit_0.eGPIO[3]; // GPIO_16bit No.0 の bit 3 を
要求
};
```

4.25.13 結合名

```
cae_name
    : IDENTIFIER
```

cae_name は、結合においては、呼び口名、または属性名である。逆結合においては、受け口名である。

4.25.14 配列添数

```
array_index
    : constant_expression
```

4.26 複合セルタイプ

複合セルタイプは、1 つ以上のセルを組合わせて新しいセルタイプを定義するものである。

複合セルタイプに組み入れられるセルを内部セルと呼ぶ。

複合セルタイプは、普通のセルタイプと同様に、呼び口、受け口、属性を持つ。

複合セルタイプに属するセルを生成するには、普通のセルタイプの場合と同様に、呼び口の結合、属性の初期値の指定を行う。複合セルタイプに属するセルの生成は、複合セルタイプの内部セルのコピーが作成されるとともに、呼び口の結合先や、属性の初期値が整えられる。

4.26.1 複合セルタイプ文

```
composite_celltype
    : 'composite' composite_celltype_name
      '{' composite_celltype_statement_list '}' ';' ;
```

複合セルタイプ文は、複合セルタイプを定義するためのものである。

複合セルタイプ文は "composite" キーワードに続く複合セルタイプ名、", " に囲まれた複合セルタイプ文リスト、最後に ";" を置く。

composite キーワードの前にセルタイプ指定子を置くことができる。

4.26.2 複合セルタイプ名

```
composite_celltype_name
    : IDENTIFIER
```

複合セルタイプ名は識別子である。

4.26.3 複合セルタイプ文リスト

```
composite_celltype_statement_list
: specified_composite_celltype_statement
| composite_celltype_statement_list
  specified_composite_celltype_statement
```

複合セルタイプ文リストは、指定子付き複合セルタイプ文のリストである。

4.26.4 指定子付き複合セルタイプ文

```
specified_composite_celltype_statement
: composite_celltype_statement
| '[' composite_celltype_statement_specifier_list ']'
  composite_celltype_statement
```

指定子付き複合セルタイプ文は、複合セルタイプ文か、 '[' , ']' で囲まれた複合セルタイプ文指定子リストに続く複合セルタイプ文である。

4.26.5 複合セルタイプ文指定子リスト

```
composite_celltype_statement_specifier_list
: composite_celltype_statement_specifier
```

複合セルタイプ文指定子リストは、複合セルタイプ文指定子である。

【制限】複数の要素に対応しない (現状では optional しかない)

4.26.6 複合セルタイプ文指定子

```
composite_celltype_statement_specifier
: 'optional'
```

複合セルタイプ文指定子 optional は、呼び口に指定できる。optional 指定子の指定は、割付けられた内部セルの呼び口における optional の指定の有無と一致させなくてはならない。

【未決定事項】複合セルタイプ文指定子として cell の allocator 指定ができるべきだが、未検討。

4.26.7 複合セルタイプ文

```
composite_celltype_statement
    : composite_port
    | composite_attribute
    | internal_cell
    | export_join
```

複合セルタイプ文は

- 複合セルタイプ口文
- 複合セルタイプ属性
- 内部セル文
- 外部結合文

のいずれかである。

複合セルタイプ文のうち、口文と複合セルタイプ属性を「インタフェース表明」と呼ぶ。残りを「複合コンポーネントセルタイプの実装」または単に「実装」と呼ぶ。

内部セルの名前のスコープは、内部セルの定義された複合セルタイプ文の内側である。

【仕様決定の理由】インタフェース表明は、通常のセルタイプと同様であり、`composite` のインタフェース表明を見るだけで、(組込まれているセルのセルタイプ定義を見なくても)セルの定義ができるようにした。

4.26.8 複合セルタイプ口文

```
composite_port
    : port
```

複合セルタイプ口文は、口文である。

4.26.9 複合セルタイプ文指定子

```
composite_celltype_statement_specifier
    : 'allocator' '(' alloc_list ')'
    | 'optional'
```

複合セルタイプ文指定子は 'allocator' または 'optional' のいずれかである。

指定子 'allocator' は内部セルに対してのみ指定できる。同じ複合セルタイプを構成する他の内部セルをアロケータセルとして指定する場合に用いる。

指定子 'optional' は呼び口に対してのみ指定できる。外部に公開するセルタイプの呼び口と 'optional' の指定の有無を一致させなくてはならない。

4.27 複合セルタイプ属性

複合セルタイプ属性は、普通のセルタイプの属性と同様に記述する。

複合セルタイプの属性は、内部セルのいずれかの属性に割付けられる。複合セルタイプの属性に与えられた初期値は、割付けられた内部セルの属性に引き継がれる。複合セルタイプに属するセルが生成される際に、複合セルタイプの属性の初期値が与えられた場合にも、同様に割付けられた内部セルの属性に引き継がれる。

4.27.1 複合セルタイプ属性文

```
composite_attribute
    : 'attr' '{' composite_attribute_declaration_list '}' ';' ;
```

複合セルタイプ属性文は、キーワード "attr" に ", " で囲まれた複合セルタイプ属性宣言リストと、末尾の ";" からなる。

4.27.2 複合セルタイプ属性宣言リスト

```
composite_attribute_declaration_list
    : attribute_declaration
    | composite_attribute_declaration_list attribute_declaration
```

複合セルタイプ属性宣言リストは、属性宣言のリストである。

宣言された属性は、内部セルのいずれかの属性から参照されなくてはならない。

属性の size.is 指定子の有無、size.is 指定子指数は、内部セルの属性と、複合セルタイプの属性で一致しなくてはならない。

4.28 内部セル

内部セル文は、複合セルタイプが内部に持つセルを定義するものである。複合セルタイプに属する生成される際には、内部セルのコピーが生成される。

4.28.1 内部セル文

```
internal_cell
    : 'cell' internal_namespace_celltype_name
internal_cell_name '{' internal_join_list '}' ';'
    | 'cell' internal_namespace_celltype_name internal_cell_name ';' ;
```

内部セル文は、”cell” キーワードに続く内部ネームスペースセルタイプ名、内部セル名、”,” に囲まれた内部結合リスト、末尾に ”;” を置く。これにより複合コンポーネントの内部のセルが定義される。

プロトタイプ宣言として、内部セル文は、”cell” キーワードに続く内部ネームスペースセルタイプ名、内部セル名を置く。プロトタイプ宣言は、受け口を相互参照しあう内部セルにおいて、後から定義される内部セルを先に定義される内部セルが参照可能にするものである。

4.28.2 内部ネームスペースセルタイプ名

```
internal_namespace_celltype_name
    : namespace_identifier
```

内部ネームスペースセルタイプ名は、ネームスペース識別子である。

4.28.3 内部セル名

```
internal_cell_name
    : IDENTIFIER
```

内部セル名は、識別子である。

4.28.4 内部結合リスト

```
internal_join_list
    :
    | internal_join_list specified_join
    | internal_join_list external_join
```

内部結合リストは、空であるか、または、指定子リスト付き結合か、セル内外部結合文か、これらを要素とするリストである。

4.28.5 セル内外部結合文

external_join

```
: internal_cell_elem_name '=>' 'composite' '.' export_name ';'
| internal_cell_elem_name '=>' export_name ';'

```

セル内外部結合文は、内部セルの呼び口を、複合セルタイプ口文で定義された複合セルタイプの呼び口に割付けて、複合セルの外側のセルの受け口に結合するためのものである。

割付記号 '=' の左辺は、内部セルの呼び口名である。

右辺は、複合セルタイプ口文で定義された呼び口の名前である。ただし、複合セルタイプ口文で定義された呼び口の名前の名前が、内部セルの持つ属性、内部変数などの名前と重複す場合には 'composite' '.' を前置して、複合セルタイプ口文で定義された呼び口であることを明示しなくてはならない。

割付ける内部セルの呼び口と複合セルタイプで定義された呼び口とで、シグニチャが一致しなくてはならない。また、内部セルのセルタイプにおいて、呼び口に optional 指定子が指定されている場合、複合セルタイプの呼び口においても optional 指定子が指定されていなくてはならない。

【使用上の注意】内部セルの属性を複合セルタイプの属性に割付けるには、指定子リスト付き結合文を使用する。内部セルの属性の割付を表す記号は '=' ではなく '=' である。

【使用上の注意】内部セルの受け口に複合セルタイプの受け口を割付けるには、外部結合文を用いる。外部結合文は、内部セルの外側に記述する。

4.29 外部結合

外部結合文は、複合セルタイプの内部セルの受け口を、複合セルタイプの受け口に割付けるためのものである

4.29.1 外部結合文

export_join

```
: export_name '=>'
    internal_ref_cell_name '.' internal_cell_elem_name ';'
| 'composite' '.' export_name '=>'
    internal_ref_cell_name '.' internal_cell_elem_name ';'

```

割付記号 '=' の左辺に、複合セルタイプの受け口名を記述する。複合セルタイプの受け口であることを明示する目的で、'composite' '.' を前置することができる。

外部結合文の割付記号 '=' の右辺には、内部セル名、'?'、内部セルの受け口名を記述する。

複合セルタイプのシグニチャと内部セルの受け口のシグニチャは一致する必要がある。なお、セルタイプの受け口に指定可能な指定子 inline は、複合セルタイプの受け口には指定できず、これを一致させる必要はない。

【仕様決定の理由】 inline は外部に対するインタフェース表明としては重要な情報ではない。

【使用上の注意】内部セルの呼び口については、セル内外部結合文によって行う。内部セルの属性は、指定子リスト付き結合文により行う。

4.29.2 外部名

```
export_name
    : IDENTIFIER
```

外部名は、単一の識別子である。

4.29.3 内部参照セル名

```
internal_ref_cell_name
    : IDENTIFIER
```

内部参照セル名は、単一の識別子である。この名前は、複合セルタイプ文の内部セル文で定義された、いずれかのセルの名前でなくてはならない。

4.29.4 内部参照要素名

```
internal_cell_elem_name
    : IDENTIFIER
```

内部参照要素名は、単一の識別子である。内部参照要素名は、内部参照セル名で指定されたセルの属するセルタイプの持つ受け口名または属性の名前でなくてはならない。

4.30 複合セルタイプ内のアロケータ

外部に結合する呼び口、受け口において send/receive 引き数を持つ関数がある場合、アロケータの指定には2通りの方法がある。

【制限】、【参照実装における制限】呼び口を内部結合する方法は存在しない。

【制限】、【参照実装における制限】 composite におけるリレーアロケータ、デバイスアロケータに対応しない。

4.30.1 外部結合する場合

外部結合する場合、通常のセルタイプにおけるアロケータの指定と同様であり、受け口側の複合セルタイプのセルでアロケータを指定する。

```
signature sSendRecv {
    /* この関数名に send, receive を使ってしまうと allocator 指
    定できない */
    ER snd( [send(sAlloc),size_is(sz)]int8_t *buf, [in]int32_t  sz );
    ER rcv( [receive(sAlloc),size_is(*sz)]int8_t **buf, [out]int32_t  *sz );
};

celltype tTestComponent {
    entry  sSendRecv eS;
    call   sSendRecv cS;
};

composite tComp {
    entry sSendRecv eSe;
    call  sSendRecv cS;

    /* 受け口のアロケータ指定子を記述しない */
    cell tTestComponent comp{
        cS => composite.cS;    /* send/receive の結合 */
    };
    composite.eSe => comp.eS; /* send/receive の結合 */
};

/* セルは、通常のセルタイプと同様に指定を行う */
[allocator(
    eSe.snd.buf=alloc.eA,
    eSe.rcv.buf=alloc.eA
)]
cell tComp comp2{
    cS = TestServ.eS;
};
```

例の中でセル TestServ の記述は通常どおりであり、例示を省略する。

4.30.2 内部結合する場合

複合セルタイプの内部にアロケータセルを持ち、内部のアロケータセルを指定することができる。この場合、通常のセルタイプのアロケータ指定とは異なり、複合セルタイプの受け口側のセルではアロケータを指定しない。

```
signature sSendRecv {
    /* この関数名に send, receive を使ってしまうと allocator 指
    定できない */
    ER snd( [send(sAlloc),size_is(sz)]int8_t *buf, [in]int32_t  sz );
    ER rcv( [receive(sAlloc),size_is(*sz)]int8_t **buf, [out]int32_t  *sz );
};

composite tComp {
    entry sSendRecv eSe;

    cell tAllocator Alloc { /* アロケータセル */
};

    [allocaotr(eSe.snd.buf=Alloc.eAlloc,
               eSe.rcv.buf=Alloc.eAlloc)]
    cell tTestComponent comp{
};
    composite.eSe => comp.eS; /* send/receive の結合 */
};

/* 通常のセルタイプの場合と異なり、受け側のセルではアロケータ指定
しない */
cell tComp comp2{
    cS = TestServ.eS;
};
```

4.31 リージョン

4.31.1 リージョンの概要

リージョンは、セルのレイアウトを制御するものである。ネームスペースの役割、すなわち一般名前有効範囲に境界を設ける役割も併せ持つ。さらにリージョン間のアクセス制御も行う。

リージョンは再帰的に設けることができる。

【補足説明】リージョンにセルのみ置くことができるのは、リージョンはセルのレイアウトを制御する機能を持つが、他のもののレイアウトを制御す

ることはできないため。逆にネームスペースは、セルのレイアウトを制御することはなく、ネームスペース内にセルを置くことができても混乱のもとである。シングニチャ、セルタイプは、コンポーネントの提供者の記述するものである。一方、セルはコンポーネントの利用者の記述するものである。それゆえ、これらが独立した名前空間に配置されることの不利はない。

4.31.2 リージョン文

```
region
    : '[' region_specifier_list ']'
      'region' region_name '{' region_statement '}' ';'
    | 'region' region_name '{' region_statement '}' ';' ;
```

リージョン文は、リージョン名と、リージョン指定子リスト、リージョン内部文を持つ。同一名のリージョンを複数に分けて指定することができる。

リージョン名が同じリージョン文を繰り返し記述することができる。ただし、リージョン指定子は、最初に現れたときに指定する必要がある。リージョン名が同じリージョン文が再び現れた場合、指定子を指定することはできない。最初に指定する必要があるのは、他のリージョンのセルからリージョン内のセルが参照される前に、参照可能かどうか決定できる必要があるためである。

リージョン名が同じリージョン文が再び現れた場合、最初に現れた場合と同じ入れ子構造としなくてはならない。

【補足説明】前方参照ができないため最初にリージョン指定子リストを指定する必要がある。現在の仕様では2回目に現れたときには、リージョン指定子を指定できない。

【補足説明】この文法のリージョンは、TECS CDL 記述では直観的にわかりやすすくない。ただし、リージョンはむやみやたらに使えばよいものではなく、またリージョン境界を設けるところは何らかの理由により独立性を高くしたい箇所であり、リージョン間の結合はそれほど多くならないはずである。

4.31.3 リージョン指定子リスト

```
region_specifier_list
    : region_specifier
    | region_specifier_list ',' region_specifier
```

リージョン指定子リストは、リージョン指定子のリストである。

4.31.4 リージョン指定子

```

region_specifier
: 'in_through' '(' plugin_name ',' plugin_arg ')'
| 'in_through' '(' ')'
| 'out_through' '(' plugin_name ',' plugin_arg ')'
| 'out_through' '(' ')'
| 'to_through' '(' namespace_region_name
                    ',' plugin_name ',' plugin_arg ')'
| 'to_through' '(' ')'
| 'node'
| 'linkunit'
| 'domain' '(' IDENTIFIER ',' STRING_LITERAL ')'
| 'class' '(' IDENTIFIER ')'

```

リージョン指定子には、`in_through`、`out_through`、`to_through`、`node`、`linkunit`、`domain`、`class` がある。

このうち `in_through`、`out_through`、`to_through` をまとめて、リージョンスルー指定子と呼ぶ。

`node`、`linkunit`、`domain`、`class` をリージョン属性と呼ぶ。

4.31.5 リージョンスルー指定子

リージョンスルー指定子は、リージョン外部のセルの呼び口からリージョン内部のセルの受け口へへ、同様に内部から外部へ、リージョン から他のリージョンへの結合を許可する。デフォルトでは、リージョンから他のリージョン（内側、外側を含む）への結合は、禁止される。

`in_through` は、リージョンの外部からリージョンの内部への結合を許可し、プラグイン (plugin) により挿入すべきスルーセルを生成させる。`in_through` は第一引数としてプラグイン名、第二引数としてプラグイン引数を取る。これらの引数が省略された場合には、リージョン外部から内部への結合を（スルーセルを生成させることなく）許可する。`in_through` は1つのリージョンに複数指定できる。複数指定された場合、順にプラグインが（指定されていれば）呼び出され、スルーセルが（プラグインにより生成されれば）挿入される。

`out_through` は、リージョンの内部からリージョンの外部への結合を許可する。その他 `in_through` と同様である。

`to_through` は、第一引数に結合可能なリージョン名、第二引数にプラグイン名、第三引数にプラグイン引数を指定する。第一引数のリージョン名は、`to_through` が読み込まれた時点では評価されない。このことによりリージョ

ンのプロトタイプ宣言は不要となる。また、リージョンのプロトタイプ宣言手段もない。to_through は、兄弟リージョン間での結合を許可する。親リージョン、子リージョン、甥リージョン、叔父リージョン、いとこリージョンなど兄弟関係にはないリージョンは指定できない。このため第一引数はリージョン名となる。

in_through, out_through, to_through で生成されるスルーセルは、受け口側のセルおよびセルの受け口が同じである場合、共有される。

【補足説明】ジェネレータの実装として、共有可能なスルーセルを共有しないオプションを設けてもよい。

4.31.6 リージョン属性

リージョン属性は、リージョンの特性を示すもので、ノード、リンク単位、ドメイン、クラスの4種類がある。TECS CDL においては、リージョンの指定子により指定する。

4.31.7 ノード (node)

ノードは、通常一つのプロセッサとメモリから構成され、1つ以上のリンク単位を持つ。ノードを超えて直接結合することはできない。

ノードを超えて、呼び口と受け口を直接結合することはできないが、コンポーネント図上、またコンポーネント記述言語においては、接続 (connect) することができる。この場合リージョン間のスループラグインにより、呼び口側、受け口側、それぞれに直接結合することなく接続させるためのコンポーネントを挿入することで実現する。実際に接続がどのように行われるかは、プラグインの仕様に依存する。

ノードが異なる場合、普通メモリ透過性がない。つまり、ポインタ値を渡したとしても、受け取った側では、ポインタ参照することができない。このため引数がポインタ型である場合、ポインタの指すメモリ領域を渡す必要がある。

例えばオペイク RPC プラグインでは、通信チャンネルを使用することで、接続することができる。オーバーヘッドが、通常の呼び出しに比べ非常に大きいため、頻繁な呼出しは適さないこと、上述のようにポインタ値を渡す場合、直接結合とは異なる振る舞いになる可能性があることに注意しなくてはならない。

TECS CDL においては、リージョンに node 指定子を指定することで、そのリージョンが一つのノードを構成することを示す。node 指定されていないリージョンは、親リージョンのノードに属する。ルートリージョンは、暗黙的に node が指定される。

OS が搭載される場合と、搭載されない場合があるが、そのいずれも TECS の対象範囲である。TECS は OS の存在を前提としない。ただし、TOPPERS 新世代カーネルとの組み合わせて使用することを想定する。

4.31.8 リンク単位 (linkUnit)

リンク単位は、リンカーにより一つのもジュールとして生成される単位である。リンク単位を超えて直接結合することはできない。

ノードの場合と同様にリージョン間のスループラグインにより異なるリンク単位のセルに接続することができる。

TECS CDL においては、リージョンに linkUnit 指定子を指定することで、そのリージョンが一つのリンク単位を構成することを示す。linkUnit 指定されていないリージョンは、親リージョンのリンクユニットに属する。

ノードは、暗黙的にリンク単位と見なされる。

4.31.9 ドメイン (domain)

ドメインは、TOPPERS/HRP2 カーネルの保護ドメインに対応させて、リージョンを保護ドメインとして扱うことを念頭に置いたものである。保護がどのように行われるかは、TECS 仕様としては規定しない。ドメインタイプごとに定義される、ドメインプラグインの仕様により規定する。また、ドメインプラグインの実装により、TOPPERS/HRP2 の保護ドメイン以外の保護機能にも対応できる。

TECS CDL においては、リージョンにドメイン指定子を指定することで、そのリージョンは、ドメインに属する。リージョン名がドメイン名となる。ドメイン指定されていないリージョンは、親リージョンのドメインに属する。

ドメイン指定子の第一引数はドメインタイプ名 (文字列ではなく、識別子) である。ドメインタイプ名により識別されるドメインタイプは、親リージョンに波及する。ただし、子リージョンから親リージョンへのドメインタイプの伝播は、ノードを超えない。一つのノードにおいては、一つのドメインタイプに制限される。ルートリージョンは、いずれのドメインにも属さない、ドメイン外 (OutOfDomain) として扱われる。従って、ドメインタイプの親リージョンへの波及は、ルートリージョンの直下のリージョンまでである。

ドメインタイプ名は、以下のようにドメインプラグイン名に変換される。

(ドメインプラグイン名) = (ドメインタイプ名) + 'Plugin'

TOPPERS/HRP2 カーネルの保護ドメインに対応させる場合、ドメインタイプを HRP2 とする。プラグイン名は HRP2Plugin となる。

4.31.10 クラス (class)

仕様未定

4.31.11 リージョン内部文

```
region_statement
:
| region_statement region_cell
| region_statement region
```

リージョン内部文は、リージョンセル文またはリージョン文を要素とするリストである。

【補足説明】リージョン内部文リストとリージョン内部文に分離されていない。

セル文は、プロトタイプ宣言と定義のいずれも可能である。ただし、セルのプロトタイプ宣言と定義の両方において、同じリージョンに属するように記述する必要がある。

4.31.12 リージョン名

```
region_name
: IDENTIFIER
```

リージョン名は、単一の識別子である。

4.31.13 ネームスペースリージョン名

```
namespace_region_name
: namespace_identifier
```

ネームスペースリージョン名は、ネームスペース識別子である。

4.31.14 リージョンセル文

```
region_cell
: cell
| '[' region_cell_specifier_list ']' cell
```

4.31.15 リージョンセル指定子リスト

```
region_cell_specifier_list
    : region_cell_specifier
    | region_cell_specifier_list region_cell_specifier
```

4.31.16 リージョンセル指定子

```
region_cell_specifier
    : 'allocator' '(' alloc_list ')'
```

4.31.17 指定リージョンのコード生成

指定されたリージョンのみコードを生成することができる。

これは、例えば分散システムで、ある部分システムのコードのみ出力することを想定したものである。

指定リージョンのコードが生成できるには、次の条件を満たさなければならない。

- 指定リージョンの境界をまたぐ結合がないこと

ただし、以下のリージョン間をまたぐスルーセルを使用した接続を持つことは可能である。

- スルーセルの内部のセルが、呼び側、受け側のいずれかのリージョンに属していること
- スルーセルの内部のセルが、リージョンをまたぐ結合を持たないこと

4.31.18 複数のシングルトンセルタイプのセル

コード生成指定されたリージョンに含まれるシングルトンセルタイプのセルを、コード生成指定されないリージョンでも定義することができる。複数のリージョンのコードを生成することが指定されている場合には、指定されたすべてのリージョンにおいて、あるシングルトンセルタイプのセルを一つだけ置くことができる。

つまり、あるシングルトンセルタイプのセルが、一つのリージョンには多くても1つに限定されるが、全体としては複数存在することができる。

4.31.19 ルートネームスペースとルートリージョン

ルートネームスペースは、いずれのネームスペース文の内側ではないところである。ルートネームにのみ構造体 `struct`, 型定義 `typedef`, 定数変数 `const` の定義を置くことができる。

ルートリージョンは、いずれのリージョン文の内側ではないところである。ルートネームスペースとルートリージョンは、共通した名前の空間を持つ。

4.32 定数定義文

4.32.1 定数定義文

```
const_statement
    : declaration
```

定数定義文は宣言文である。

定数定義文は、以下のいずれかの型の定数を定義するものである。

- 整数型
- 浮動小数型
- ブール型
- ポインタ型

ポインタ型の定数としては、関数へのポインタを扱うことはできない。

【補足説明】構造体定数、配列定数は定義できない。この制限は、参照実装で実現できたものを仕様としているため。それ以外の理由はない。

宣言文の初期化子は、定数式または `C_EXP` である。

ポインタ型の場合、初期化子の定数式は、整数値をポインタ型にキャストした値に限られる。

【補足説明】& 演算子によって、他の変数などのポインタ値を得ることはできない。

【参照実装における制限】定数は、`define` によるプリプロセッサマクロとして定義される。このため、C 言語の定数は通常記憶域が割付けられるが、TECS では記憶域が割付けられない。

【参照実装における制限】定数は、`define` によるプリプロセッサマクロとして定義される。このため属性や変数などの名前と重複すると、C 言語のコンパイル段階で、これらが定数と置換されて思わぬ文法エラーを引き起こす。

4.33 宣言

宣言文は、定数定義文、属性の定義、内部変数の定義において用いられる。宣言文で定義する上記のものは、初期化子により初期値を与えることができる。

4.33.1 宣言文

```
declaration
    : type_specifier_qualifier_list init_declarator_list ';' ;
```

【補足説明】K&Rとの違い: storage class が指定できない、型が省略できない

4.33.2 宣言指定子

```
declaration_specifiers
    : type_specifier
    | type_qualifier type_specifier
```

4.33.3 初期化宣言子リスト

```
init_declarator_list
    : init_declarator
    | init_declarator_list ',' init_declarator
```

4.33.4 初期化宣言子

```
init_declarator
    : declarator
    | declarator '=' initializer
```

4.34 初期化子

4.34.1 初期化子リスト

```
initializer_list
    : initializer
    | initializer_list ',' initializer
```

4.34.2 初期化子

```
# assignment_expression を constant_expression に変更
initializer # mikan
    : constant_expression
    | '{' initializer_list '}'
    | '{' initializer_list ',' '}'
    | 'C_EXP' '(' STRING_LITERAL ')'
```

初期化子は、定数式、`''` で囲まれた初期化子リストまたは `C_EXP` 初期化子である。

このいずれを取りうるかは、初期化される変数の型による。

定数式は、整数型、浮動小数型、ブール型、ポインタ型を初期化することができる。ただし、`size_is` 指定されたポインタ型を除く。

`''` で囲まれた初期化子は、構造体型、配列型、`size_is` 指定されたポインタ型を初期化することができる。

【参照実装における制限】`size_is` 指定された構造体へのポインタ型は初期化子を指定できない。結果として `var` にのみ用いることができる。

`C_EXP` 初期化子は、初期化する変数が集成型（構造体型、配列型）でない場合に、初期化子として指定することができる。

4.34.3 C_EXP 初期化子

`C_EXP` 初期化子は、文字列リテラルを引数に取る。引数の文字列リテラルは、ジェネレータの出力の `C` 言語初期化子として出力される。ヘッダファイルで `define` 定義される値を参照するために使用することが意図されている。

文字列リテラルは、名前置換が行われる。ファクトリにおける名前置換と同じ規則である。

4.34.4 composite における C_EXP の名前置換

`composite` における名前置換は、特別な規則が適用される。`composite` の `attr` に現れる `C_EXP` における `id`, `ct`, `$cell$` の名前置換では、複合セルタイプの名前、複合セルの名前に置換される。他の名前置換は、複合セルタイプの内部セルが展開されてコピーされたセルの名前によって置換される。

以下に、複合セルタイプにおける名前置換の例を示す。次のような TECS CDL 記述があるものとする。ここで `tCelltype`, `tCelltyp2` の定義は省略する。

```
composite tComposite {
    attr {
        int32_t a = C_EXP( "A_$id$" );
```

```

};
cell tCelltype Cell1 {
    a = composite.a;
    b = C_EXP( "B_$id$" );
};
cell tCelltype2 Cell2 {
    a = composite.a;
    c = C_EXP( "C_$id$" );
};
};

cell tComposite CompositeCell {
};

```

CompositeCell における、名前置換の結果は A_\$id\$ は "A_CompositeCell" に、B_\$id\$ は "B_CompositeCell_Cell" に、C_\$id\$ は "C_Composite_Cell2" となる。属性 a は Cell1, Cell2 とともに、同じ初期値を持つことになる。

【仕様決定の理由】この仕様は、タスクとタスク例外を別のセルとして生成し、それらを composite で一つのセルにまとめる際に、タスクの ID の名前を一致させるために導入された（他の用途でも使用してもよい）

4.34.5 シグニチャプラグイン (generate) 文

シグニチャプラグイン文は、シグニチャに対しプラグインを適用することを指示するものである。

```

signature_plugin
    : GENERATE '(' plugin_name ',' namespace_signature_name ',' plugin_arg ')' ';

```

シグニチャプラグイン文は 'generate' キーワードに続いて、'(', ')' に囲まれた3つの引数、末尾に ';' を置く。

シグニチャプラグインを呼び出す記述は、シグニチャ定義とは独立に行う。

第一引数は、プラグイン名である。単一の識別子を引数として与える。

第二引数は、ネームスペースシグニチャ名である。

第三引数は、プラグインオプションである。プラグインオプションは、文字列リテラルを引数として与える。

シグニチャプラグインの対象となるシグニチャは、シグニチャプラグインを適用する前に定義されていないとしない（シグニチャの前方参照は、できない）。

【補足説明】シグニチャプラグインは、スループラグインを実現する基礎として用いることができる。シグニチャプラグインに、スループラグインが

適用された呼び口に対応づくシグニチャを適用して、スループラグインで挿入するセルのセルタイプおよびセルタイプコードを生成させる。スループラグインではそのセルタイプのセルを生成させる、という実現が想定される。

【補足説明】シグニチャプラグインでは、シグニチャを変形することを想定していない。この点で、セルプラグインやセルタイププラグインとは異なる。シグニチャプラグイン文をシグニチャの定義と分離する所以である。

第5章 TECS コンポーネント実装

TECS コンポーネント実装は、TECS コンポーネントの振る舞いのコードを記述する方法を規定するものである。

5.1 定義

5.1.1 セルタイプコード

セルタイプコードは、コンポーネントの振る舞いを記述するものであり、受け口関数の実装である。

5.1.2 セルタイプヘッダ

セルタイプコードからインクルードすべき、TECS ジェネレータの生成したヘッダファイル。

5.1.3 グローバル名

グローバル名は、セルタイプコードの記述において用いられ、ネームスペース名を含む一意の名前である。グローバル名は、セルタイプ名の前に、それらの属するネームスペース名にアンダスコア '_' を付加したものを前置したものである。

(グローバル名) = (ネームスペース名) + '_' + (セルタイプ名)

ここで + は文字列の連結を表す。

ネームスペースが親ネームスペースに属する場合、親ネームスペース名にアンダスコア '_' を付加したものを、更に前置する。ルートネームスペースに属する場合には、ネームスペース名を含まない。

【補足説明】ネームスペース名、セルタイプ名、シグニチャ名が '_' を含む場合でも、名前付け規則に従った名前を付与していれば、名前衝突を回避できるはずである。

【補足説明】セルのグローバル名は、セルタイプコードの中で用いられることはない。

【補足説明】シグニチャのグローバル名についても、TECS ジェネレータで生成されるヘッダファイル内で用いられるだけで、セルタイプコードで記述することはない。

5.1.4 インライン受け口関数

受け口関数が、インラインと指定された受け口のものが、そうでないかを区別する必要がある場合、以下のように呼び分ける。インラインと指定された受け口の受け口関数を、インライン受け口関数と呼ぶ。同様にインライン指定されていない受け口の受け口関数を、非インライン受け口関数と呼ぶ。

5.2 セルタイプコードに関する規則

5.2.1 使用言語

セルタイプコードは C 言語で記述する。

【補足説明】C++ として記述することもできる。ただし、外部名を C 言語に合わせる。

5.2.2 記述単位

セルタイプコードは、セルタイプごとにファイルを分けて記述する。

一つのセルタイプのセルタイプコードを、複数のファイルに分けて記述してもよい。

インラインの受け口関数と非インラインの受け口関数とは、ファイルを分けて記述する。

【補足説明】複数のセルタイプのセルタイプコードを一つにまとめることはできない。

5.2.3 ファイル名

非インライン受け口関数を格納するファイルの名前は、セルタイプのグローバル名に C 言語の拡張子 .c を付加したものとする。C++ で記述する場合には、適切な拡張子を付加する。セルタイプコードを複数のファイルに分けて記述する場合、他のファイル名を付与する（名前付け規則は定めない）。

以下に、セルタイプ名が tCelltype である場合のファイル名の例を示す。

```
tCelltype.c
```

5.2.4 インラインのセルタイプコードのファイル名

インラインの受け口関数を格納するファイルのファイル名は、セルタイプのグローバル名に `'_inline.h'` を付加したヘッダファイルとする。インラインの受け口関数を実装するセルタイプコードを複数のファイルに分けて記述する場合、セルタイプのグローバル名に `'_inline.h'` を付加した名前のヘッダファイルからインクルードする。

以下に、セルタイプのパス名が `nNS::tCelltype` である場合のファイル名の例を示す。

```
nNS_tCelltype_inline.c
```

5.2.5 ファイルの記述内容

セルタイプコードを格納するファイルには、以下を記述する。

- セルタイプヘッダファイルのインクルード
- 受け口関数
- 受け口関数から呼び出される関数

5.2.6 セルタイプヘッダのインクルード

非インライン受け口関数を実装するセルタイプコードを記述するファイルでは、セルタイプヘッダをインクルードしなくてはならない。以下にセルタイプ `tCelltype` の場合の例を示す。

```
#include "tCelltype_tecsgen.h"
```

セルタイプコードを記述しようとするセルタイプ以外のセルタイプヘッダをインクルードしてはならない。

【補足説明】複数のセルタイプのセルタイプコードを一つのファイルに含めることは想定されていない。C 言語のマクロを駆使して関数の結合先をコンパイル時に決定する TECS の実装では、やむを得ない制限である。

5.2.7 参照できるもの

セルタイプコードの中では、以下のものを参照できる。

これらは、いずれもセルタイプコードを記述しようとするセルタイプの持つものでなくてはならない。

- 呼び口関数

- 属性
- 内部変数

この他に、ライブラリとして提供されるものを参照することができるが、コンポーネントとして提供されるものの代わりにライブラリ参照するのは、望ましくない。

これ以外のものを参照する場合、逸脱になる。

5.2.8 受け口関数の形式

受け口に対応付けられたシグニチャで定義されたすべての関数を受け口関数として記述する。

受け口関数の名前は、以下の通りである。

(受け口関数名) = (受け口名) + '_' + (シグニチャで定義された関数名)

【補足説明】受け口関数名は、マクロにより、受け口関数のグローバル名に置換される。

セルタイプが非シングルトンの場合、シグニチャで定義された関数ヘッダに比べ、第一引数が挿入される。挿入される第一引数の型は、CELLIDX 型である。

以下に受け口関数の形式 (関数ヘッダ) の例を示す。

TECS CDL の記述 (非シングルトンの場合)

```
signature sSignature{
  ER func1( [in]int32_t inval, [out]int32_t *outval );
  ER func2( [in,size_is(size)]const int8_t *buf, [in]int32_t size );
};
celltype tCelltype {
  entry sSignature eEntry;
};
```

tCelltype のセルタイプコードに記述する必要のある受け口関数

```
ER eEntry_func1( CELLIDX idx, int32_t inval, int32_t *outval )
ER eEntry_func2( CELLIDX idx, const int8_t *buf, int32_t size )
```

なお CELLIDX 型が何であるかは、ここでは規定しない。ポインタ値であったり整数値であったりする。

5.2.9 受け口関数の形式（受け口配列の場合）

受け口配列の場合、第二引数に配列添数を挿入する。

次に受け口配列の例を示す。以下のような TECS CDL の記述があったとする。

```
signature sSignature{
  ER func1( [in]int32_t inval, [out]int32_t *outval );
  ER func2( [in,size_is(size)]const int8_t *buf, [in]int32_t size );
};
celltype tCelltype {
  entry sSignature eEntry[2];
};
```

tCelltype のセルタイプコードに記述する必要のある受け口関数の関数ヘッダは、以下ようになる。

```
ER eEntry_func1( CELLIDX idx, int_t subscript, int32_t inval, int32_t *outval )
ER eEntry_func2( CELLIDX idx, int_t subscript, const int8_t *buf, int32_t size )
```

5.2.10 CB ポインタ

CB ポインタは、非シングルトンのセルタイプの場合に、セルを選択するために必要である。シングルトンのセルタイプでは、CB ポインタを得ることはできない。

以下、セルタイプが非シングルトンの場合の、受け口関数の中で CB ポインタを得る方法を説明する。CB ポインタが何物であるかは、ここでは規定しない。

CB ポインタを得るコードの例を以下に示す。

```
CELLCB *p_cellcb; /* p_cellcb の名前を変えてはならない */
if (VALID_IDX(idx)) {
  p_cellcb = GET_CELLCB(idx);
}
else {
  return(E_ID);
}
```

以下に必須の要件を記す。

- 非シングルトンセルタイプの場合、CB ポインタを得る
- CB ポインタの変数名は p_cellcb とする

- CB ポインタの型は `CELLCB` 型とする
- 第一引き数 `CELLIDX idx` を検査する関数(マクロ)として `VALID_IDX` を使用する
- 第一引き数 `CELLIDX idx` を CB ポインタに変換する関数(マクロ)として `GET_CELLCB` を使用する

先ほどの CB ポインタを得るコードの例にあって、この要件にないのは、`VALID_IDX` で `idx` が不正と判断された場合に `E_ID` を返すことである。TOPPERS/ASP 系の OS では `E_ID` を返すのが妥当であるが、戻り値の型が `ER` や `ER_INT` でない場合、あるいは TOPPERS/ASP 系以外の OS で動作させることを目的に記述している場合には、`E_ID` を返す必要はない。

【補足説明】実際の実装において `idx.is_id` が指定されていない場合、`VALID_IDX` が `false` を返すことはない。

【補足説明】呼び口関数、属性、内部変数のいずれも参照しない場合、`p_cellcb` は非参照となる。

5.2.11 呼び口関数

セルタイプコードにおいて、呼び口のシグニチャで定義された関数を呼び出し可能である。呼び口関数は、シグニチャで定義された関数と以下の点で異なる。

(呼び口関数名) = (呼びけ口名) + '_' + (シグニチャで定義された関数名)

以下に例を示す。以下のような TECS CDL の記述があったとする。

```
signature sSignature{
    ER func1( [in]int32_t inval, [out]int32_t *outval );
    ER func2( [in,size_is(size)]const int8_t *buf, [in]int32_t size );
};
celltype tCelltype {
    call sSignature cCall;
};
```

`tCelltype` のセルタイプコードでの呼び口関数は以下ようになる。

```
ER cCall_func1( int32_t inval, int32_t *outval )
ER cCall_func2( const int8_t *buf, int32_t size )
```

【補足説明】受け口関数の場合は、第一引き数 `idx` が挿入されたが、呼び口関数では挿入されない。

5.2.12 呼び口関数（呼び口配列の場合）

次に呼び口配列の例を示す。以下のような TECS CDL の記述があったとする。

```
signature sSignature{
  ER func1( [in]int32_t inval, [out]int32_t *outval );
  ER func2( [in,size_is(size)]const int8_t *buf, [in]int32_t size );
};
celltype tCelltype {
  call sSignature cCall[];
};
```

tCelltype のセルタイプコードでの呼び口関数は以下ようになる。

```
ER cCall_func1( int_t subscript, int32_t inval, int32_t *outval )
ER cCall_func2( int_t subscript, const int8_t *buf, int32_t size )
```

非配列の場合に比べ、第一引き数に配列添数が加えられる。配列添数の最小値は 0 である。最大値は、マクロ N_CP_cCall (cCall は呼び口名に置き換える) により知ることができる。

5.2.13 属性

セルタイプコードにおいて、属性を参照可能である。

属性参照名は、以下のように属性名に 'ATTR_' を前置きしたものである。

(属性参照名) = 'ATTR_' + (属性名)

以下に例を示す。以下のような TECS CDL の記述があったとする。

```
celltype tCelltype {
  attr {
    int32_t attribute;
  }
};
```

tCelltype のセルタイプコードでの属性参照は以下ようになる。

```
ATTR_attribute
```

ATTR_attribute は左辺値として扱うことができる。

5.2.14 内部変数

セルタイプコードにおいて、内部変数を参照可能である。

内部変数参照名は、以下のように内部変数名に 'VAR_' を前置きしたものである。

(内部変数参照名) = 'VAR_' + (内部変数名)

以下に例を示す。以下のような TECS CDL の記述があったとする。

```
celltype tCelltype {
  var {
    int32_t variable;
  }
};
```

tCelltype のセルタイプコードでの内部変数参照は以下ようになる。

VAR_variable

VAR_variable は左辺値として扱うことができる。

5.2.15 非シングルトンセルタイプの場合のセルタイプコードの例

これまでの、セルタイプコードの規則に従ったコードの例を示す。ここでは、非シングルトンセルタイプの場合を示す。

```
signature sSignature{
  ER func1( [in]int32_t inval, [out]int32_t *outval );
  ER_INT func2( [in,size_is(size)]const uint8_t *buf, [in]int32_t size );
};
celltype tCelltype {
  entry sSignature eEntry;
  attr {
    int32_t attribute;
  };
  var {
    int32_t variable;
  };
};
```

tCelltype のセルタイプコードは以下ようになる。

```

ER eEntry_func1( CELLIDX idx, int32_t inval, int32_t *outval )
{
    /* CB ポインタを得るコード */
    CELLCB *p_cellcb;          /* p_cellcb の名前を変えてはなら
    ない */
    if (VALID_IDX(idx)) {
        p_cellcb = GET_CELLCB(idx);
    }
    else {
        return(E_ID);
    }

    *out_val = inval - ATTR_attribute; /* 属性 attribute を参照

    return E_OK;
};
ER_INT eEntry_func2( CELLIDX idx, const uint8_t *buf, int32_t size )
{
    /* CB ポインタを得るコード */
    CELLCB *p_cellcb;          /* p_cellcb の名前を変えてはなら
    ない */
    int32_t i, sum = 0;
    if (VALID_IDX(idx)) {
        p_cellcb = GET_CELLCB(idx);
    }
    else {
        return(E_ID);
    }

    for( i = 0; i < size; i++ )
sum += buf[i];

    return sum;
};

```

【補足説明】ATTR_attribute は CB ポインタ p_cellcb を含むマクロであることを想定するが、限定するものではない。実例として、ATTR_attribute ではさらに以下のものを含む。属性は ROM に置かれるが、これを INIB と呼ぶ。CB から INIB へのポインタ参照も ATTR_attribute マクロに含まれる。これは、最適化状態において変わりうる。

【補足説明】VAR_variable は CB ポインタ p_cellcb を含むマクロであることを想定するが、規定するものではない。

5.2.16 シングルトンセルタイプの場合のセルタイプコード

これまでの、セルタイプコードの規定に従ったコードの実例を示す。ここでは、シングルトンセルタイプの場合を示す。

```
signature sSignature{
    ER func1( [in]int32_t inval, [out]int32_t *outval );
    ER_INT func2( [in,size_is(size)]const uint8_t *buf, [in]int32_t size );
};
[singleton]
celltype tCelltype {
    entry sSignature eEntry;
    attr {
        int32_t attribute;
    };
    var {
        int32_t variable;
    };
};
```

tCelltype のセルタイプコードは以下のようになる。

```
ER eEntry_func1( int32_t inval, int32_t *outval )
{
    *out_val = inval - ATTR_attribute; /* 属性 attribute を参照

    return E_OK;
};
ER_INT eEntry_func2( const uint8_t *buf, int32_t size )
{
    int32_t i, sum = 0;

    for( i = 0; i < size; i++ )
sum += buf[i];

    return sum;
};
```

【補足説明】シングルトンセルタイプのコードでは、idx 引き数がない、CB ポインタを得るコードがない点で、非シングルトンセルタイプのコードと異なる。

【補足説明】ATTR_attribute は INIB 構造体名を含むマクロであることを想定するが、規定するものではない。

【補足説明】VAR_variable は CB 構造体名を含むマクロであることを想定するが、規定するものではない。

5.3 初期化コード

TECS のコードが実行される前に、以下のマクロを呼び出して初期化を行う。

```
INITILIZE_TECS()
```

INITILIZE_TECS は、必要に応じて以下を行う。

- 内部変数の初期化
- CB から INIB へのポインタ設定

初期化を行う前に TECS のコードが実行された場合の振る舞いは、未定義である。

【補足説明】TECS ジェネレータ V1.2.* まで INITIALIZE_TECSGEN であったが、V1.3.0.0 以降、INITIALIZE_TECS に変更となった。この変更に対応していない場合、初期化が行われない。

【補足説明】TECS ジェネレータ V1.2.* まで、INITIALIZE_TECS() は TECS ジェネレータにオプション -R が指定された時だけ生成されていたが、V1.3.0.1 以降、常に生成される。ただし -R が指定されていない場合、INITIALIZE_TECS() の内容は空であり INITIALIZE_TECS() が呼び出されない従来のコードも動作する。ただし、将来においても成り立つとは限らない。

5.4 FOREACH_CELL マクロ

セルタイプコードにおいて、そのセルタイプに属するすべてのセルの変数を操作を行うために FOREACH_CELL マクロを使用できる。主として、初期化の際に用いる。

5.4.1 FOREACH_CELL マクロの使用

すべてのセルに対する操作を行うには、FOREACH_CELL マクロで始め END_FOR_EACHCELL で終わるループにより実現できる。以下にコードの例を示す。

```

#include "tCelltype_tecsgen.h"

...

func()
{
    /* tCelltype のすべてのセルについて初期値を作業変数に移す */

    CELLCB *p_cellcb;          /* 短縮形で属性、内部変数を参照す
    するために p_cellcb とします */
    int i;                      /* ループ変数を用意する必要があります
    . 名前は適当で構いません */

    FOREACH_CELL(i,p_cellcb) /* FOREACH_CELL でループの開始を宣言
    します */
        VAR_a0 = ATTR_a;      /* 短縮形 VAR_a0, ATTR_a で変数、属性
    参照できます */
        VAR_b0 = ATTR_b;
    END_FOREACH_CELL
}

```

5.4.2 FOREACH_CELL マクロの多重使用

FOREACH_CELL マクロを多重ループで用いることができる。この場合、内側のループ内では外側のループのセルの属性、変数には直接アクセスすることはできない。外側のループの属性、変数は別の自動変数に写し取ること、内側のループで参照できる。

```

#include "tCelltype_tecsgen.h"

...

func()
{
    CELLCB *p_cellcb;          /* 短縮形で属性参照するために p_cellcb と
    しました */
    int i;

    FOREACH_CELL(i,p_cellcb)
        CELLCB *p;            /* p としたので短縮形で属性参照で
    きません */
        int j; /* 内側のループ変数を j とします */

```

```

    FOREACH_CELL(j,p)
        /* 外側のループには短縮形が使えるが、内側のループはセルタイ
        プのグローバル名を伴うマクロを使用 */
        if ( ATTR_a == tCelltype_ATTR_a( p ) ) {
            ...
        }
    END_FOREACH_CELL
END_FOREACH_CELL
}

```

5.5 ファクトリ

ファクトリでは、任意のファイルに出力できるが、以下の2つのファイルは、予め想定するファイルである。

5.5.1 コンフィギュレーションファイル

ファクトリの第一の使用目的は、コンフィギュレーションファイルに TOP-PERS/ASP 系カーネルの静的 API を生成することである。TECS ジェネレータにより生成されるコンフィギュレーションファイルの名前は、以下を推奨する。

```
tecsgen.cfg
```

以下は、コンフィギュレーションファイルに静的 API を生成する TECS CDL の記述例である。セルが生成されるごとに、コンフィギュレーションファイルに静的 API が出力される。

```

celltype tTask {
    attr {
        ID                id = C_EXP( "TASKID_$id$" );
        VP_INT            exinf;
        [omit]ATR         tskatr;
        [omit]PRI         itskpri;
        [omit]SIZE        stksize = 4096;
    };
}

factory {
    write( "tecsgen.cfg",
        "CRE_TSK(%s,{%s,&$id$_CB,tTask_start_task,%s,%s,NULL});",
        id, tskatr, itskpri, stksize );
}

```

```
};  
};
```

【補足説明】ファクトリで記述できることは限られている。判断や繰り返しが必要な場合、セルタイププラグインにより実現する。

5.5.2 ファクトリヘッダ

ファクトリヘッダは、セルタイプコードに取込むためのヘッダファイルであり、セルタイプヘッダにおいてインクルードされる。ファクトリヘッダには、セルの属性、変数において C_EXP で与えられた初期値に含まれるマクロの定義を記述することができる。ファクトリヘッダの名前は、以下のとおりである。CELLTYPENAME の部分は、セルタイプ名に置き換えられる。

```
CELLTYPENAME_factory.h
```

以下の例は、セルタイプファクトリで TOPPERS/ASP のコンフィグレータの生成する kernel_cfg.h をインクルードするものである。これによりコンフィグレータの出力するマクロを、セルの属性、変数の C_EXP 初期化子の中で参照することができる。

```
celltype tCelltype {  
    FACTORY {  
        write( "$ct$_factory.h", "#include \"kernel_cfg.h\"" );  
    };  
};
```

付録 A 名前付け規則

名前付け規則は、文法の一部ではなく、慣用的なものである。

A.1 接頭文字の規則

要素	接頭文字	その他	例	備考
シグニチャ	s	2文字目は大文字	sMotor, sLamp	タスクコンテキスト
シグニチャ	si	3文字目は大文字	siTask	非タスクコンテキスト (non-task)
シグニチャ	sn	3文字目は大文字	snInit	カーネル外コンテキスト
関数	なし	先頭は小文字	rotateRight	
引き数	なし	先頭は小文字	goalPositionX	
セルタイプ	t	2文字目は大文字	tMotor, tLamp	
セル	なし	先頭文字大文字	Motor, Lamp	
呼び口	c	2文字目は大文字	cMotor	
受け口	e	2文字目は大文字	eMotor	
属性	なし	先頭は小文字	countLimit	
内部変数	なし	先頭は小文字	upCount	

【参照実装における制限】定数は、上記の規則にはないが、属性、内部変数、関数の名前と重複しないこと。定数はヘッダファイルで `define` によって定義されるため、定数と他のものの名前が重複すると、C コンパイラでコンパイルするときに、分りにくいエラーが発生する。

A.2 単語区切り

- 単語の1文字目は、大文字とする
- 単語と単語は直接連結する（`'`などを置かない）

A.3 型情報

通常、型に関する情報を名前に含めないが、以下の場合には名前に含める。

- 属性、変数、引き数がポインタ型の場合、'p_'を前置する
- 属性、変数、引き数がパッケージの場合、'pk_'を前置する
- 属性、変数、引き数がパッケージへのポインタ型の場合、'ppk_'を前置する

付録B マクロ

B.1 マクロ一覧

以下に、TECS ジェネレータが生成し、セルタイプコードで使用可能なマクロの一覧を記す。セルタイプコードを記述する場合、短縮形を用いる。

マクロ	短縮形	通常形
IDX の正当性チェック	VALID_IDX	tCelltype_VALID_IDX
セル CB を得るマクロ	GET_CELLCB	tCelltype_GET_CELLCB
属性アクセスマクロ	ATTR_attribute	tCelltype_ATTR_attribute
内部変数アクセスマクロ	VAR_variable	tCelltype_VAR_variable
呼び口関数マクロ	cCall_func	
受け口関数マクロ	eEntry_func	tCelltype_eEntry_func
呼び口関数マクロ	cCall_func	
呼び口配列サイズマクロ	N_CP_cCall	
FOREACH_CELL マクロ	FOREACH_CELL	

この表では、一例を示している。以下のような置き換えが必要である。

- attribute は属性名に置き換える
- variable は内部変数名に置き換える
- func は関数名に置き換える
- cCall は呼び口名に置き換える
- eEntry は受け口名に置き換える
- tCelltype はセルタイプ名に置き換える

B.2 短縮形マクロ

短縮形マクロは、通常形に優先して使用されることが意図されている。また、テンプレートコードは、短縮形の使用を意図して生成されている。属性・変数参照マクロでは CELLCB へのポインタが p_cellcb という名前で定義されることが仮定されている。

- セル CB を得るマクロ (短縮形)

セル CB を得るマクロは、GET_CELLCB に固定である。

マクロ定義例

```
#define GET_CELLCB(idx)  tAttribute_GET_CELLCB(idx)
```

- IDX の正当性チェックマクロ (短縮形)

IDX の正当性チェックマクロは VALID_IDX に固定である。

マクロ定義例

```
#define VALID_IDX(IDX)  tAttribute_VALID_IDX(IDX)
```

- 属性アクセスマクロ (短縮形)

属性アクセスマクロは、接頭辞 'ATTR_' に属性名を結合した名前である。

マクロ定義例

```
#define ATTR_size          ((p_cellcb)->_inib->size)
#define ATTR_size_array   ((p_cellcb)->_inib->size_array)
#define ATTR_ptr           ((p_cellcb)->_inib->ptr)
```

- 内部変数アクセスマクロ (短縮形)

内部変数アクセスマクロは、接頭辞 'VAR_' に属性名を結合した名前である。

マクロ定義例

```
#define VAR_sz_array      ((p_cellcb)->sz_array)
```

- 呼び口配列の大きさを得るマクロ (短縮形のみ)

呼び口配列の大きさを得るマクロは、接頭辞 'N_CP_' に呼び口名を結合した名前である。呼び口が配列の場合のみ、このマクロが生成される。

マクロ定義例

```
#define N_CP_carray      (2)
```

- オプション呼び口テストマクロ (短縮形)

呼び口配列の場合、このマクロで結合をチェックする前に、呼び口配列の大きさが 1 以上であることを確認すること。

マクロ定義例

```
#define is_cCall_joined  ((p_cellcb)->_inib->cCall!=0)
```

B.3 通常形マクロ

通常形のマクロは、他のセルの属性、変数を参照するために使用することが意図されている。

- IDX の正当性チェックマクロ

```
#define tAttribute_VALID_IDX(IDX) (1)
```

- セル CB を得るマクロ */

```
#define tAttribute_GET_CELLCB(idx) (idx)
```

- 属性アクセスマクロ */

```
#define tAttribute_ATTR_size( p_that ) ((p_that)->_inib->size)
#define tAttribute_ATTR_size_array( p_that ) ((p_that)->_inib->size_array)
#define tAttribute_ATTR_ptr( p_that ) ((p_that)->_inib->ptr)

#define tAttribute_GET_size(p_that) ((p_that)->_inib->size)
#define tAttribute_GET_size_array(p_that) ((p_that)->_inib->size_array)
#define tAttribute_GET_ptr(p_that) ((p_that)->_inib->ptr)
```

- 変数アクセスマクロ

```
#define tAttribute_VAR_sz_array ((p_cellcb)->sz_array)

#define tAttribute_GET_sz_array(p_that) ((p_that)->sz_array)
#define tAttribute_SET_sz_array(p_that,val) ((p_that)->sz_array=(val))
```

- オプション呼び口テストマクロ

呼び口配列の場合、このマクロで結合をチェックする前に、呼び口配列の大きさが 1 以上であることを確認すること。

マクロ定義例

```
#define tCelltype_is_cCall_joined(p_that) ((p_that)->_inib->cCall!=0)
```


付録C ファイルの一覧

TECS 仕様のファイルの一覧をまとめたものである。各ファイルの詳細は、それぞれの説明のある章を参照のこと。

C.1 ソースコードファイルの一覧

種類	ファイル名	自動生成	備考
セルタイプコード	CELLTYPE.c		テンプレート自動生成
セルタイプヘッダ	CELLLTYPE_tecsgen.h		
セルタイプファクトリヘッダ	CELLLTYPE_factory.h		
セルタイプ tecsgen コード	CELLLTYPE_tecsgen.c		
セルタイプインラインコード	CELLLTYPE_inline.h		テンプレート自動生成

セルタイプコードは、`inline` 指定されていない受け口関数を実装するものである。一方、セルタイプインラインコードは `inline` 指定された受け口関数を実装するものである。

自動生成されたファイルは、編集することは意図されていない。テンプレートは、TECS ジェネレータの実行により上書きされるため、別に移してから編集する。

付録D TECS ジェネレータの出力

ここでは、TECS ジェネレータの出力するものについて説明する。

1. CB と INIB
2. 初期化コード
3. 互換性のモデル

D.1 CB と INIB

D.1.1 CB と INIB

CB と INIB は、個々のセルの情報を記憶するメモリ領域である。セルごとに、セルの CB 領域、INIB 領域が存在する（最適化により、存在しない場合はありうる）。CB はセルごとに RAM 確保される領域である。INIB は、セルごとに ROM に確保される領域である。

この呼称は TOPPERS/ASP の実装において用いられているもので、カーネルオブジェクトのデータを記憶する領域に与えられたものである。TECS において、同じ呼称を、同様な意味で引き継いでいる。CB と INIB の名称や、主な配置は次表の通りである。

呼称	英語の名称	日本語の名称	所在	標準的な内容
CB	Control Block	コントロールブロック	RAM	var, CB から INIB へのポインタ
INIB	Initialize Block	初期化ブロック	ROM	attr, 呼び口ディスクリプタ

呼び口ディスクリプタとは、呼び口が結合された受け口に関する情報である。

D.1.2 CB と INIB のコード

CB と INIB は、TECS ジェネレータにより、セルタイプ `tecs`gen コードとして、セルタイプごとに以下の名前のファイルにまとめて出力される。ただし `tCELLTYPE` はセルタイプ名に置き換える。

```
tCELLTYPE_tecsgen.c
```

D.1.3 CB と INIB のバリエーション

前項で CB, INIB の標準的な内容を示したが、以下のようなバリエーションが考えられる。

1. var, CB から INIB へのポインタを CB に、attr, 呼び口ディスクリプタを INIB に配置する
2. var, attr, 呼び口ディスクリプタ、すべてを CB に配置する
3. 1. に対し、呼び口ディスクリプタを CB に配置する
4. 1. に対し、INIB から CB へのポインタを INIB に配置する

1. は前項の標準的な内容をであり、TECS ジェネレータのデフォルトである。

2. は、補助記憶装置から RAM 上にダウンロードして動作させるタイプの組込みシステムに向いている。汎用 OS 上での動作にも適している。TECS ジェネレータに実装されていて、-r オプションで、これに対応するコードが生成される。

3. は、呼び出しオーバーヘッドを低減する目的である。ただし、少し RAM の使用量が増加する。関数呼び出しに伴う CB から INIB への参照が減る。また、一版に ROM のアクセスは RAM に比べ遅いため、RAM に置かれた CB のみにアクセスした方が処理速度を多少向上できる。(関数テーブルや、実行コードが ROM に置かれているのであるから、このことは、それほど寄与しないはずである)

4. は、RAM の使用量を低減するものである。ただし、多少実行速度は増加する。

【補足説明】3., 4. については、TECS ジェネレータに実装されていない。

D.2 初期化コード

ここで扱う初期化コードとは、TECS の RAM 領域、すなわち CB を初期化するコードのことである。

D.2.1 初期化コードが必要となるケース

内部変数 var は必ず、読み書き可能な RAM 上に置かなくてはならない。内部変数には、初期値が与えられている場合がある。内部変数が置かれるメモリ領域を、セルの CB と呼ぶが、どのように CB の領域に、内部変数の初期値を設定するかが、問題となる。さらには CB から INIB へのポインタも設定する必要がある。

セルの CB は初期化領域におく方法と、未初期化領域に置く方法が考えられる。

初期化領域に置く場合は、ここでの初期化の対象外であるが、ここに補足する。初期化領域に置いた場合、汎用 OS 上で動作させるのであれば、補助記憶装置から RAM に初期化領域のデータのコピーが作られ、初期化が完了する。ROM 化システムでは、ROM 上に、初期化領域、すなわちセルの CB のデータを持ち、起動時に RAM 上の本来のアドレスにコピーする。しかし、0 に初期化する場合も少なくないのに、すべてのデータを持つ必要があり、ROM に占めるセルの CB のデータが増えてしまい、効率が悪い。

未初期化領域に置く場合、セルの CB を生成するための C 言語のコードとしては、初期値を与えないで置く。このため、起動時にセルの CB に初期値を設定しなくてはならない。

【補足説明】TECS ジェネレータにおいて -R を指定する場合に、初期化コードが必要となる。

D.2.2 初期化コードの役割

前提条件として、少なくとも TECS の CB がレイアウトされる領域の RAM は、0 にクリアされるものとする。

初期化コードは、以下のことを行う。

- RAM 上の CB から ROM 上の INIB へのポインタの初期化
- RAM 上に置かれる var の初期化

TECS ジェネレータは、RAM の初期化コードを生成する。

TECS ジェネレータは標準では、ROM から RAM へコピーするのに適したコードが生成される。このコードは、汎用 OS においては適切なものである。しかし、ROM に RAM へコピーする方式では、RAM を初期化するのに必要以上のメモリを必要とする。

D.2.3 初期化コードの生成と利用

TECS ジェネレータは、セルの CB を初期化するための初期化コードを生成する。初期化コードは、コンポーネントのコードが動作する前に呼び出され、初期化されなければならない。

D.2.4 初期化マクロ

初期化マクロは、セルタイプヘッダに含まれる。

```
SET_CB_INIB_POINTER(i,p_cb)
INITIALIZE_CB(p_cb)
```

SET_CB_INIB_POINTER は、CB から INIB へのポインタをセットする。
INITIALIZE_CB は、var を初期化する。ただし、初期値の指定されていない変数は初期化しない。

RESET_CB INITIALIZE_CB の初期化に加え、初期値の指定されていない変数をクリアする。

【参照実装における制限】RESET_CB INITIALIZE_CB は生成されない。

SET_CB_INIB_POINTER を最初に呼び出さなくてはならない。デフォルトでは CB には var、INIB には attribute が配置される。このため INITIALIZE_CB で attribute を参照して var を初期化する場合、不正アクセスとなる。

D.2.5 初期化プログラム

セルタイプのセル CB を初期化するプログラムは、以下の名前で生成される。このプログラムは tCELLTYPE_tecsgen.c に出力される。ここで tCELLTYPE はセルタイプ名である。

```
tCELLTYPE_CB_initialize()
```

D.2.6 全体初期化マクロ

全体初期化マクロとして、以下が生成される。

```
INITILIZE_TECS()
```

【使用上の注意】参照実装の TECS ジェネレータにおいて -R オプションを指定してコード生成した場合、TECS のアプリケーションプログラムが動作を開始する前に、必ず上記のマクロを呼び出す必要がある。

(V1.3.0.1 以降) -R 指定にかかわらず INITILIZE_TECS() が出力されるので、常に呼び出す。

```
INITIALIZE_TECS();
```

(V1.3.0.0 まで) -R を指定していない場合、INITIALIZE_TECS マクロは生成されない。-R の有無のいずれにも対応できるようにするためには、以下のようにする。

```
#ifndef TECS
    INITIALIZE_TECSEGEN();
#endif
```

D.3 データ構造のモデル

呼び元の CB から呼び先の CB や、呼び先の関数に至るまでのデータ構造を示す。

このデータ構造は、すべてのデータを CB にのみ置き、INIB は存在しない場合のものである。

【制限】最も標準的なケースである CB と INIB の両方を持つ場合が記されていない。【未決定事項】詳細な説明。

構造図の構成要素のうち、受け口ディスクリプタ、受け口スケルトン関数については、後の項で説明する。

D.3.1 基本データ構造図

図 D.1 に、基本データ構造図を示す。

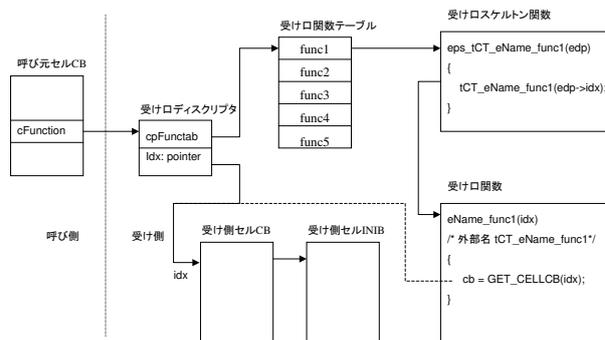


図 D.1: 基本データ構造図

D.3.2 idx_is_id の場合のデータ構造図

図 D.2 に、idx_is_id の場合のデータ構造図を示す。

D.3.3 受け口配列の場合のデータ構造図

図 D.3 に、受け口配列の場合のデータ構造図を示す。

D.3.4 呼び口配列の場合のデータ構造図

図 D.4 に、呼び口配列の場合のデータ構造図を示す。

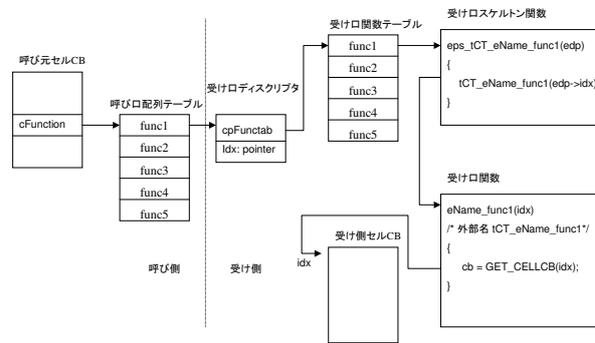


図 D.4: 呼び口配列の場合のデータ構造図

`idx_is_id` でない場合、`IDX` はセル CB へのポインタである。`idx_is_id` でない場合、`IDX` の正当性チェックマクロ `VALID_IDX` は、常に `true` を返す。また `GET_CELLCB` マクロは、`IDX` の値をそのまま CB へのポインタとして返す。

`idx_is_id` の場合、`IDX` はセルを識別する整数値の ID である。`idx_is_id` の場合、`IDX` の正当性チェックマクロ `VALID_IDX` は、ID の値の有効性を検査する。また `GET_CELLCB` マクロは、`IDX` の値からセル CB へのポインタに変換する。

受け口が、受け口配列の場合、受け口ディスクリプタは、3 つ目の要素として、受け口配列の添数を持つ。

【仕様決定の理由】受け口ディスクリプタが受け口配列の添数を保持するため、呼び元は、呼び先の受け口が配列であっても、なくても受け口ディスクリプタへのポインタを保持するだけでよい。あるセルタイプに属する複数セルの呼び先の受け口が、受け口配列のものとそうでないものの両方があったとしても、セルタイプコードは一樣の処理で対応できる。

受け口ディスクリプタは、受け口スケルトン関数とともに、セルタイプ `tecsген` コードとして、セルタイプごとに以下の名前のファイルに出力される。ただし、`tCELLTYPE` はセルタイプの名前に置き換える。

```
tCELLTYPE_tecsgen.c
```

D.3.7 受け口スケルトン関数

上述の図を構成する、受け口スケルトン関数について説明する。

受け口スケルトン関数も、TECS ジェネレータによって自動生成されるものである。受け口スケルトン関数は、セルタイプの受け口の各関数ごとに生成される。

受け口スケルトン関数は、第一引き数に受け口ディスクリプタへのポインタを取る。第二引き数以降は、シグニチャで定義された関数の引き数である。

受け口配列の場合であっても、配列添数は関数の引き数には加えられない。受け口ディスクリプタの3つ目の要素から取り出す。

受け口スケルトン関数は、受け口ディスクリプタから `IDX` を取り出して、受け口関数を呼び出す。受け口関数は、セルタイプコードとしてセルの振舞いが記述されたものである。

【仕様決定の理由】このような実装は、オーバーヘッドが大ききようにも思える。C++ のように `CB` に関数テーブルへのポインタを埋める方式も考えられる。しかし `TECS` では最適化により受け口ディスクリプタや受け口スケルトン関数が省略される場合が少なくないため、必ずしもオーバーヘッドにならない。また、複数の受け口を持っていても、全ての受け口を対等に扱うことができる。受け口スケルトン関数が存在するもう一つの理由は、検証機構を埋め込むためである。

D.4 呼び口、受け口お最適化

セルタイプの呼び口、受け口について、いくつかの条件を満たすとき、最適化を実施する。これにより、条件によって、まったくオーバーヘッドがなくなる。

条件により以下の最適化が行われる。

呼び口最適化

- 単一セル最適化
- 関数テーブル不要最適化
- 受け口ディスクリプタ不要最適化

受け口最適化

- 関数テーブル不要最適化
- 受け口ディスクリプタ不要最適化

D.4.1 呼び口最適化の条件

以下は、あるセルタイプのある呼び口に対する条件である。

- 単一セル最適化

その呼び口の、すべての結合先が同じセルの同じ受け口である場合に行う。

- 関数テーブル不要最適化

その呼び口の、すべての受け口が同じセルタイプの同じ受け口である場合に行う。

- 受け口ディスクリプタ不要最適化

関数テーブル不要最適化と同じ条件に加え、呼び先のセルの受け口が受け口配列でない場合に行う。

D.4.2 呼び口最適化実施内容

- 単一セル最適化 (@b_cell_unique)

- セル CB/INIB に呼び口情報を置かない
- 呼び口情報 (受け側セルの IDX) を、呼び口関数マクロに埋め込む

- 関数テーブル不要最適化 (@b_VMT_useless)

• 呼び口関数マクロは、受け口関数または受け口スケルトン関数を関数テーブルを介することなく直接呼び出す

- 受け口ディスクリプタ不要最適化 (@b_skelton_useless)

• 呼び口関数マクロは、受け口関数を関数テーブルを介することなく直接呼び出す

(関数テーブル不要最適化における呼び口関数マクロが直接呼び出す先が決定される)

• セル CB/INIB に埋め込まれる、呼び先の受け口ディスクリプタへのポインタの代わりに、呼び先セルの IDX を置く

D.4.3 結合状態と呼び口最適化実施内容

呼び口の結合先の受け口が単一のセルタイプの一つの受け口に限られる場合、呼び口最適化が可能になる。

- 1) 呼び先があるセルの単一の受け口

呼び先のセルが一つで、単一の受け口である場合、その呼び口について以下の最適化を行う。

単一のセル最適化

呼び口の情報を CB/INIB に置かない (@b_cell_unique)

受け口が配列ではない

受け口関数を VMT を介することなく呼び出す (第一引数は IDX)

```
(@b_skelton_useless, @b_VMT_useless)
```

受け口が配列である

受け口スケルトン関数を呼び出す (第一引数は受け口ディスクリプタ)
(@b_VMT_useless)

2) 呼び先があるセルタイプの単一の受け口

あるセルタイプのある呼び口について、すべてのセルの呼び先が (あるセルタイプの) 同じ受け口であって、呼び先のセルが複数ある場合、その呼び口について以下の最適化を行う。

受け口が配列ではない

受け側がシングルトンではない

CB には、受け口ディスクリプタへのポインタの代わりに呼び先の IDX

```
(@b_cell_unique==false && @b_skelton_useless)
```

なお、呼び口配列の場合には呼び口 VDES 配列の記憶するポインタを IDX に置き換える

受け口関数を直接呼び出す (@b_skelton_useless, @b_VMT_useless)

受け口が配列である

CB に受け口ディスクリプタへのポインタを置く (非最適化時と同じ)

受け口スケルトン関数を呼び出す (@b_VMT_useless)

【補足説明】@b_cell_unique, @b_skelton_useless, @b_VMT_useless は参照実装における変数名を表す。受け口ディスクリプタ不要最適化のフラグ名が @b_skelton_useless であるのは、受け口ディスクリプタが不要のときは、受け口スケルトン関数も不要となるため。

D.4.4 呼び口最適化 (呼び口配列)

【制限】未実施

D.4.5 受け口最適化

- 関数テーブル不要最適化
- 受け口ディスクリプタ不要最適化

ある受け口に結合されるすべての呼び口が、関数テーブル不要最適化、受け口ディスクリプタ不要最適化されている場合、受け口の最適化が可能となる。

D.4.6 受け口最適化実施内容

- 関数テーブル不要最適化 (@b_VMT_useless)

受け口の関数テーブルを生成しない

受け口ディスクリプタに含まれる関数テーブルへのポインタを NULL に初期化

- 受け口ディスクリプタ不要最適化 (@b_skelton_useless)

受け口ディスクリプタを生成しない

受け口スケルトン関数を生成しない

D.4.7 受け口最適化 (受け口配列)

【制限】未実施

D.4.8 attribute 最適化

すべての cell の attribute が同じ値の場合、ATTR_attr を定数に define

【制限】未実施

D.5 CB、INIB の最適化のモデル

D.5.1 CB、INIB の基本的なデータ配置

ROM/RAM サポートの場合 CB と INIB に分けて出力される。ただし、条件により、いずれか、または両方が省略される可能性がある。CB は読み書き可能な RAM に INIB は ROM に配置されることが仮定される。

CB, INIB へ置かれる情報は、以下のように分けられる。

1)CB に置かれる情報

- 内部変数 (var) . ただし size_is 指定されたポインタを除く
- INIB へのポインタ

2)INIB に置かれる情報

• 呼び口に結合された受け口を示す受け口ディスクリプタへのポインタ
最適化により、受け側の CB へのポインタになったり省略されることがある

- 属性 (attr) . ただし omit 指定されたものを除く
- 内部変数 (var) の size_is 指定された変数 (ポインタ)

これらの情報はセルタイプの定義に依存し、存在しない場合もある。

D.5.2 CB が存在しない場合

size_is 指定されていない内部変数 (var) が存在しない場合、CB は作成されない。この場合 GET_CELLCB() マクロ関数は、INIB へのポインタを返す。INIB へのポインタは INIB が存在しない場合には省略される。

一方 omit 指定されていない属性も、size_is 指定された変数 (ポインタ) も、最適化により省略されていない呼び口の結合先の受け口ディスクリプタも存在しない場合には INIB は省略される。

もし CB も INIB も省略された場合には GET_CELLCB は NULL ポインタを返す。この場合受け口ディスクリプタポインタを取り出して結合先を呼出したり、属性や変数を参照することはなく、このポインタを介してアクセスするものは何もない。

D.5.3 CB、INIB の最適化

a) CB も INIB も存在する場合: has_INIB? && has_CB?

RAM 節約にはなるが CB へのポインタからたどるので、アクセスが遅くなる。

速度優先の場合には CB に呼び口へのポインタを配置することもできる。

【参照実装における制限】上記は、未実装

- CB

CB に INIB へのポインタを _inib の名前でもつ

- 呼び口呼出しマクロ

```
#define tCt2_cCall1_func( p_that, in, out ) \
(p_that)->_inib->cCall1->VMT->func( \
(p_that)->_inib->cCall1, (in), (out) )
```

(p_that) から _inib を介した呼び出しとなる

- 属性アクセスマクロ

```
#define ATTR_a ((p_cellcb)->_inib->a)
```

b) CB のみの場合: !has_INIB? && has_CB?

= attr も、呼び口も、存在しない場合

INIB へのポインタは生成しない。

c) INIB のみの場合: has_INIB? && !has_CB?

= var が存在しない場合

CB の型を INIB の型に置換する。

CB へのポインタは、直接 INIB を指す。

```
#define tag_CelltypeName_CB tag_CelltypeName_INIB
```

attribute a に対する出力は以下ようになる

```
#define ATTR_a ((p_cellcb)->a)
```

```
#define tCt2_cCall1_func( p_that, in, out ) \
(p_that)->cCall1->VMT->func( \
(p_that)->cCall1, (in), (out) )
```

d) INIB, CB ともに存在しない場合: !has_INIB? && !has_CB?

CB へのポインタを引くことができないので、以下ようになる

```
#define tCt2_GET_CELLCB(idx) (0)
```

D.6 Makefile

TECS ジェネレータの生成する Makefile には以下の 3 種類がある。

- Makefile.templ ... テンプレート：そのまま Makefile として使用するのに必要なほとんどが含まれる
- Makefile.tecsgen ... gen ディレクトリに生成されるファイルおよびセルタイプコードのリストの変数を定義する
- Makefile.depend ... tecsgen の入出力および中間ファイルの依存関係を示す

【制限】現在実装では Makefile.depend の依存関係には import_C から読み込むファイルは含まれない

D.6.1 Makefile.tecsgen の内容

ファイルのリスト

TECS_IMPORTS import, import_C により取込まれたヘッダ（自身を含む）

TECS_COBJS	TECSGEN_COBJS PLUGIN_OBJS CELLTYPE_COBJS の総和
TECSGEN_COBJS	tCELLTYPE_tecsgen.o のリスト
CELLTYPE_COBJS	セルタイプコードの .o のリスト
PLUGIN_COBJS	プラグイン生成された .c から作成される .o のリスト
TECSGEN_SRCS	tCELLTYPE_tecsgen.c のリスト
PLUGIN_SRCS	プラグイン生成された .c のリスト

D.6.2 Makefile.depend の内容

依存関係 1 : TECS CDL ファイルとその出力の依存関係
被依存ファイル :

- 引数で指定される TECS CDL ファイル
- import で読み込まれる TECS CDL ファイル
- import_C で読み込まれる C ヘッダファイル

依存ファイル :

- 出力される C ファイル (含ヘッダファイル)
- factory で出力されるファイル

tCELLTYPE_tecsgen.c : abc.cdl abc.h

依存関係 2 : TECS CDL ファイルの出力とその include するヘッダファイルの依存関係

被依存ファイル :

- sSIGNATURE.h
- tCELLTYPE_factory.h
- tCELLTYPE_tecsgen.h

依存ファイル :

- CELLTYPE_tecsgen.c
- CELLTYPE_templ.c

.c が依存する .h ファイルは、完全にはわからない。factory により include される .h ファイルは、cpp を通して、さらに include されるものをチェックする必要がある。しかし、define, include パスが決定できないと、cpp を通すことができない。

D.6.3 Makefile.templ の内容

C コンパイルコマンド

```
CC = gcc
CFLAGS = -D ... -I ...

.c.o :
$(CC) $(CFLAGS) -o $@ $<
```

リンクコマンド

```
TARGET = 1st_CDL.exe
OBJECTS = $(CELLTYPE_TEMPL_CODE) $(CELLTYPE_TECSGEN)

GEN_DIR = gen
CELLTYPE_TECSGEN = $(GEN_DIR)/CELLTYPE_tecsgen.o...

CTC_DIR = src

move_template_ctc :
```

D.7 互換性

D.7.1 互換性

ここで扱う互換性とは、TECS コンポーネントを再利用する際に問題とならないようにすることである。

互換性については、以下の2通りのレベルが考えられる。

- ソースコード互換性
- バイナリコード互換性

ソースコード互換性は、すべてのソースコードを閲覧可能で、すべてのソースコードをコンパイルしてアプリケーションをビルドする場合に、重要である。

一方、バイナリ互換性は、コンポーネントをバイナリ形式で流通させる場合に重要である。例えばソースコードを開示せずリロケータブルオブジェクトの形でコンポーネント提供したい場合や、ロードブルモジュールの実現である。

この他にオペイクな RPC においてもバイナリレベルの互換性が重要になるが、その点については、RPC プラグインの仕様として規定される。

互換性モデルとして重要なことは、再利用性が妨げられないようにすること、開発したソフトウェア資産の価値が低下しないようにすることである。

汎用 OS 上での再利用性では、実行可能な形式での配布が前提であり、バイナリ互換性が重要になる。一方、組込みシステムにおいては、装置に動的にモジュールを取込む場合を除けば、予めシステムにプログラムを組み込んでおくため、バイナリ互換性よりもソースコード互換性が重視される場合が少なくない。また、ソースコードを修正することなく他の環境へポーティングできれば、ソースコードの資産価値を低下させないですむ。

このような考察のもと、ソースコード互換性を TECS 仕様における必須の要件とする。

D.7.2 ソースコード互換性

上述のように TECS 仕様におけるソースコード互換性は、必須の要件である。しかし、すべてのソースコードの互換性が必須ではない。

ソースコードの中で、ハンドライティングされるものの互換性が必須である。ツールによって自動に生成されるコードは互換性がなくても、ツールによって生成しなおすために必要な労力はきわめて小さい。

ハンドライティングされるソースコードは、セルタイプのソースコードである。TECS 仕様に準拠したツール類を開発する場合、セルタイプコードの互換性が損なわれないようにしなくてはならない。

D.7.3 バイナリコード互換性

【未決定事項】バイナリコード互換性は未検討である。

組み込みシステムにおいて、ソフトウェア部品がバイナリモジュールとして提供される場合がないわけではない。ソースコード非公開で販売される組み込み用のソフトウェア部品も少なからず存在する。また、ロードブルモジュールのように、出荷された装置にモジュールを動的に組み込みたい要求もある。このためバイナリコード互換性も必要である。