Ⅱ－１．シナリオ関数は３つのプログラム問題を解決する「プログラム革命」技術

MTI

■　シナリオ関数はなぜプログラム革命？

１．特許化されたシナリオ関数の構造は実行様相を可視化

２．シナリオ関数は「バグゼロ」を担保する開発技術

３．シナリオ関数は「デッドロック」を担保する開発

４．シナリオ関数は「ウイルス無力化の仕組み」を担保する開発

実践例：シナリオ関数はプログラムの全てのデータ経路を可視化

■シナリオ関数で使用される用語集

© 2017 MTInternational Co,Ltd.

■　シナリオ関数はなぜプログラム革命？

プログラムとは何か　：プログラムは業務処理で，電子機器製品に内蔵されて電子機器全般で使用されている。

『　プログラムはあらゆる電子機器の稼働にとっての必須技術である。　』

プログラムの役割　：プログラムはデータ生成することを目的としている。

『　電子機器の稼働とは：入力処理⇒演算・データ移動処理等⇒出力処理（画面，帳票，ファイル更新）の実行である。』

①入力処理とは：外部からのデータ（情報）を入力命令の入力データ領域（受信領域）に入力（受信）させます
②演算（含む移動）処理とは：入力データを元に出力に必要とするデータに演算命令等で加工します。
③出力処理とは：演算処理結果データをを出力命令で画面・帳票・ファイル更新として出力します。

プログラム開発方法の現状　：プログラムを使用する電子機器は安全・安心の担保をして社会に提供する義務を伴います。

『　20世紀以来，プログラム開発は[潜在バグは仕方がない］をIT業界はもちろん社会全体が受け入れている。』

21世紀の電子機器製品とは　：プログラムを使用する電子機器は安全・安心の担保をして社会に提供する義務を伴います。

電子機器の課題

『　電子機器の高度な機能（俗説的なAI）でのエラーは人体に直接的影響を及ぼすリスクが非常に高まっている。』

電子機器に求められる必須技術

『　安全に稼働すると担保（100%完全なプログラム）し，安心して使用できる電子機器製品の提供ができるかである』

安全/安心の電子機器とは

『　稼働中に停電以外，システムダウン・エラーを起こさないプログラム（潜在バグのない）を搭載した電子機器製品である』

潜在バグのないプログラム開発技術がプログラム革命

潜在バグのないプログラム提供

『　シナリオ関数のプログラム定義構造』で開発されたプログラムは：従来プログラム開発方法の3大課題を解決する。　』

①【潜在バグのないバグゼロプログラム】を　担保する開発技術である。
②【デッドロックを起こさないプログラム】を　担保する開発である。
③【コンピュータウイルスをプログラム自身で無力化するプログラム】を　担保する開発である。

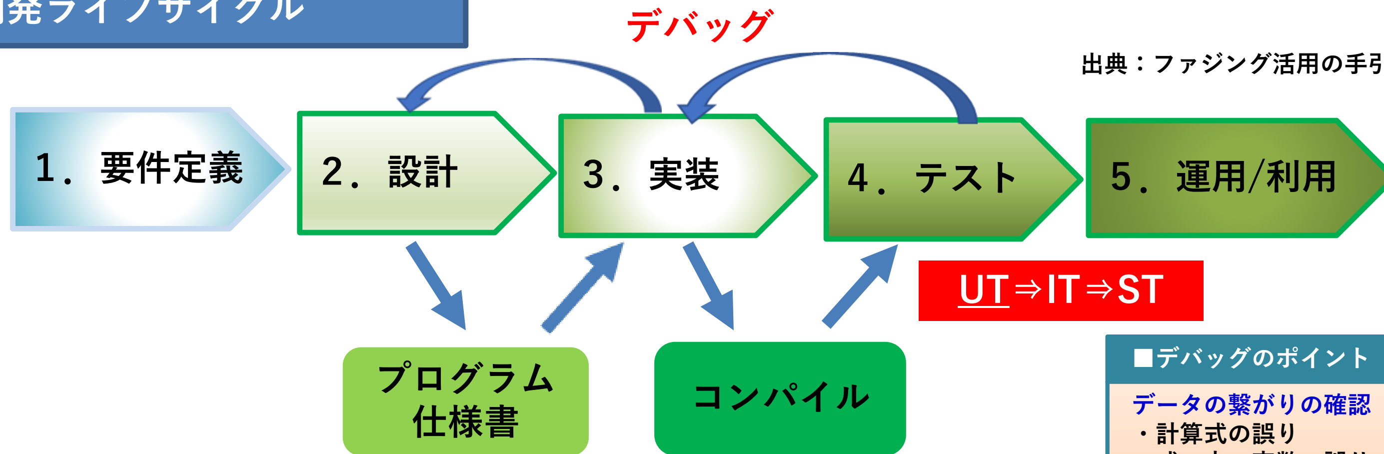
© 2017 MT　International Co . Ltd.

■　プログラム開発方法の現状　VS　シナリオ関数の開発とは

プログラム開発方法の現状

『　［潜在バグは仕方がない］をIT業界はもちろん社会全体が受け入れている。』

開発ライフサイクル



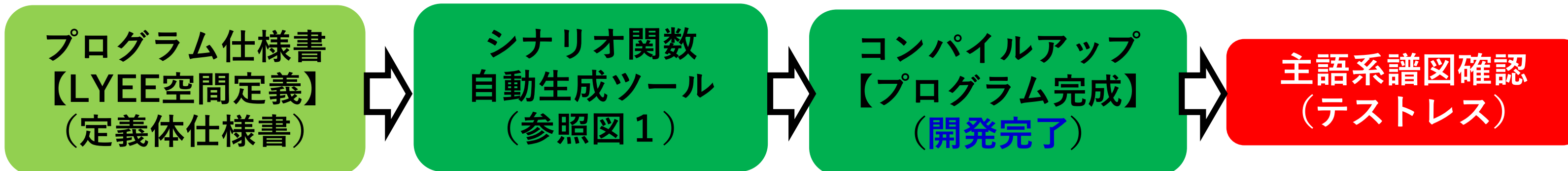
■デバッグのポイント

データの繋がりの確認

・計算式の誤り
・式の中の変数の誤り
・代入先の変数の誤り
・判断条件の誤り
・分岐先の誤り
・ループのさせ方の誤り
・文や計算式の順序の誤り

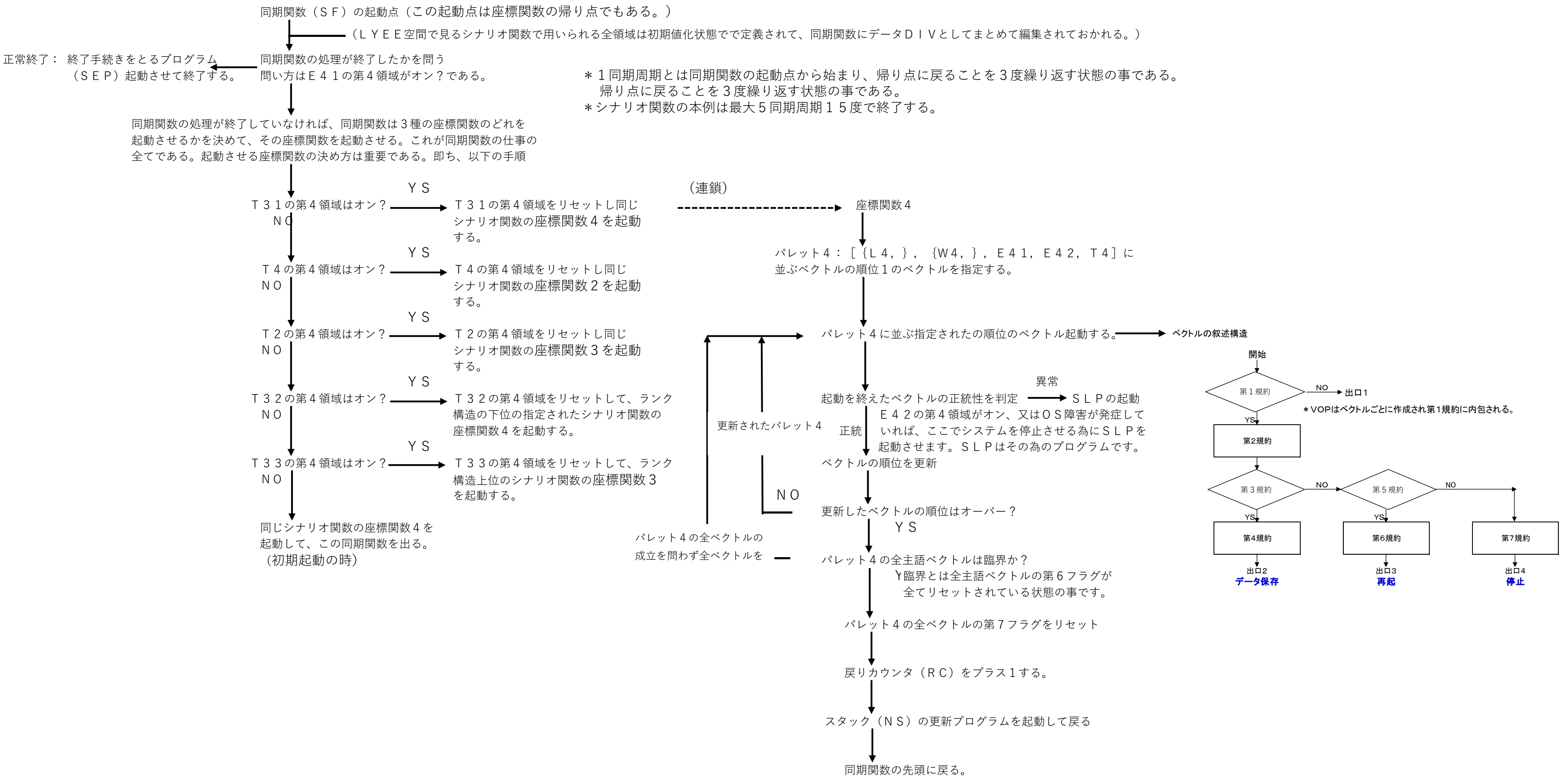
シナリオ関数の開発

要件定義⇒設計段階でシナリオ関数定義体仕様書作成を前提に推進



© 2017 MT　International Co . Ltd.

参照図１．同期関数・座標関数・ベクトルプログラム開発の基本



1．特許化されたシナリオ関数の構造は実行様相を可視化

■シナリオ関数と従来プログラムの違いとは：講演１．の３項の補完

１．シナリオ関数の構造でプログラムの内容を把握できる。：プログラムを経営資産として把握できる。

①シナリオ関数の定義式とはプログラムの意味である解（S)を数式で表現する

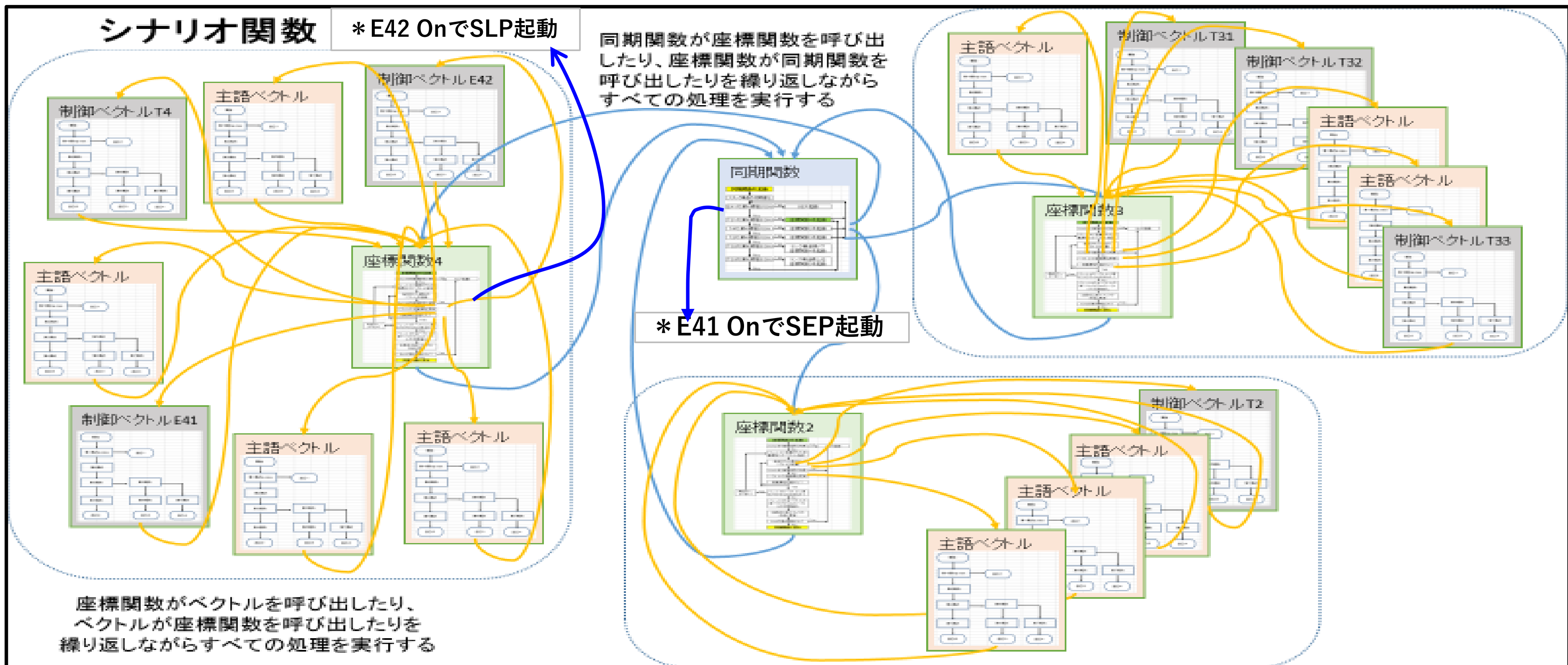
S＝φ0(φ4[{L4},{W4},T4,E41,E42］＋φ2[{R2},{L2},T2］＋φ3[{L3},T31,T32,T33］)

②全てのデータ生成に関与する全ての命令文は機能に従い１命令文単位でベクトル化されます。：ベクトルが実行単位になる(スレッド)
データ生成に関与する命令文のベクトルとは L4：代入文，W4：出力文，L2：定値・定置文，R2：入力文，L3：条件文
ベクトルは７つの規約（開発規則）で構成され，規約には普遍性があるのでロボット生産を可能にする。

③ OS機能の役割を担う制御ベクトルとしてＴ４，Ｔ２，Ｔ３１，Ｔ３２，Ｔ３３，Ｅ４１，Ｅ４２

④シナリオ関数の仕組は
実行全体を統括する同期関数：φ０，ベクトルの実行順序を統括する座標関数：φ４，φ２，φ３，ベクトルで構成される。

２．シナリオ関数の実行様相



© 2017 MT　International Co .Ltd.

２．シナリオ関数は「バグゼロ」を担保する開発技術

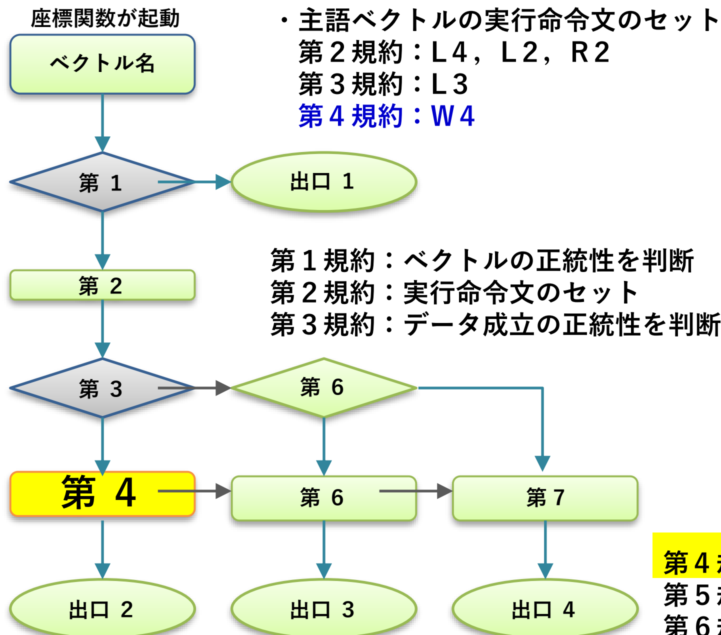
「バグゼロ」技術とは：データ生成に関与する1命令文の存在証明ができる。

■シナリオ関数における命令文の存在証明とは

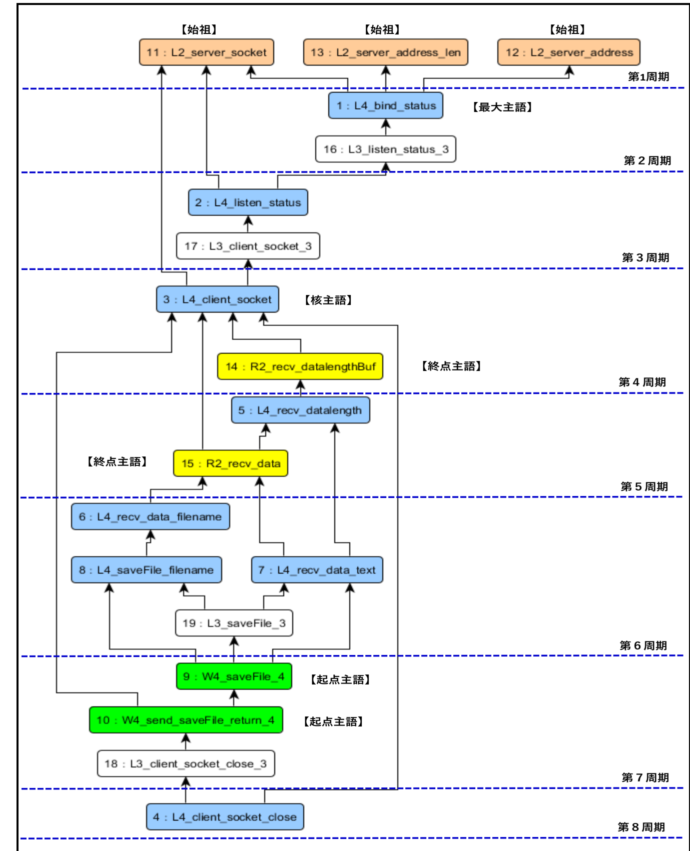
１．データ生成に関与する命令文は全て主語ベクトル化する
２．主語ベクトルは7個の規約で存在証明をする
３．第3規約で命令文の変数主語の存在証明を担保する
４．主語ベクトルの第3規約で主語系譜図を生成できる
５．主語系譜図はシナリオ関数の解（S)を可視化する

■ベクトルの７つの規約の役割

（最小単位命令文のプログラム定義構造）



■主語系譜図サンプル例

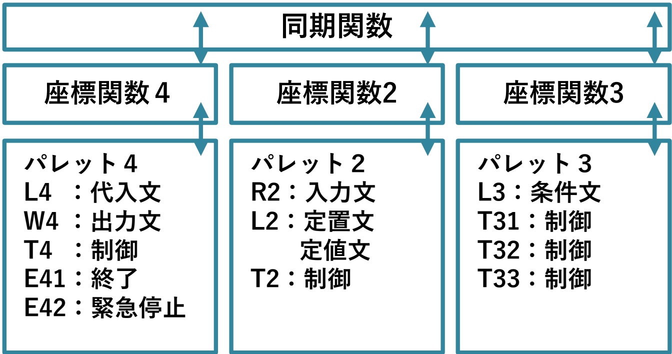


第４規約：成立データの保存領域
第５規約：ベクトルの再起及び停止指示の判断
第６規約：再起指示（同じ座標周期で成立可能性有り）
第７規約：停止指示(成立可能性無し；座標関数移動)

© 2017 MT　International Co .Ltd.

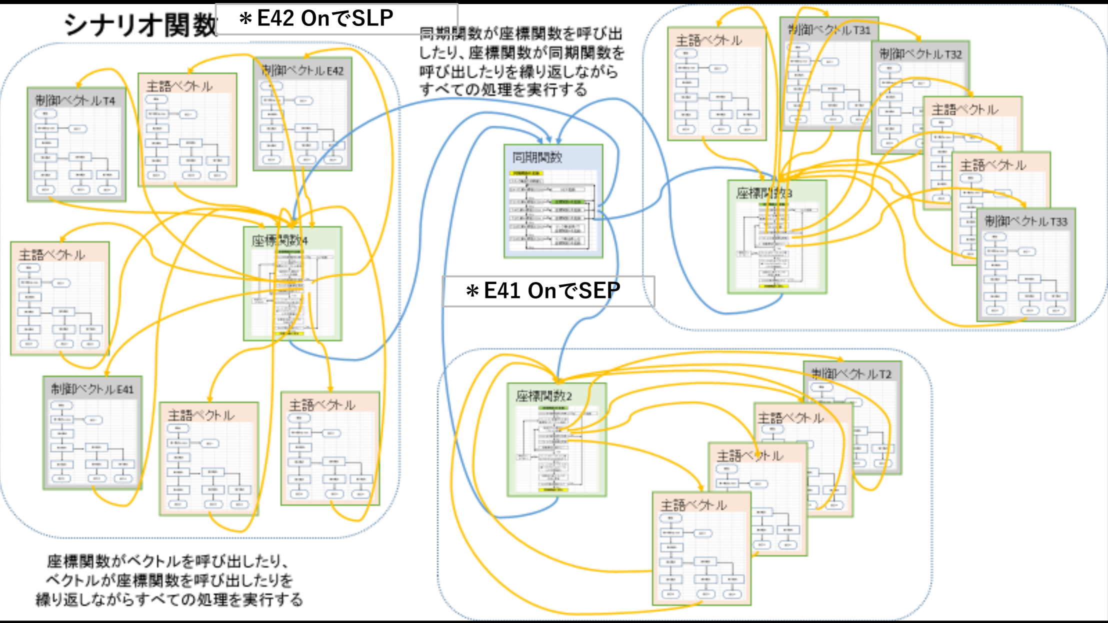
3．シナリオ関数は「デッドロック」を意識しない開発技術

「デッドロック」の解消技術とは：データ処理順序はシナリオ関数の仕組みが担保する



1．シナリオ関数の実行順序は
基本はパレット上のベクトルの繰り返し処理でデータを生成する
同期関数⇒座標関数４⇒ベクトル⇒座標関数４⇒同期関数⇒
座標関数２⇒ベクトル⇒座標関数２⇒同期関数⇒
座標関数３⇒ベクトル⇒座標関数３⇒同期関数⇒
座標関数４⇒ベクトル（制御ベクトルE41ON）⇒座標関数４⇒
同期関数（E４１ONの確認）⇒SEPを起動させる
＊System Ending Program：終了プログラム

2．主語ベクトルの実行順序は座標関数の呼び出しで実行処理し
出口２，３，４で座標関数を呼び出す仕組みになります
＊ベクトルの7つの規約の役割を参照



■最新ハードウェア（マルチコア）の活用が可能
最先端技術をソフトウェア技術が追随できる
1．主語ベクトルがスレッド単位になる
2．座標関数がマルチコアに対応する数量だけ
主語ベクトルをセットする
3．主語ベクトルは実行処理後座標関数を
呼び出し次の実行待ち主語ベクトルを起動

■緊急停止後復元処理プログラムが容易に開発
1．E42ベクトルが緊急停止を指示する。
・主語成立数の増加が止まる：処理のループ
2．座標関数４はE42ベクトルの第4規約「ON」
3．座標関数４はSLPを起動する。
＊Soft Landing Program：復元プログラム
4．全てのシナリオ関数にE42が搭載されます。
5．SLP開発はシナリオ関数処理の明確化で，
緊急停止後の復元方法も予測可能になる。

© 2017 MT　International Co .Ltd.

4.シナリオ関数は「ウイルス無力化の仕組み」を担保する開発技術

ウイルスの無力化とは：ウイルスデータ侵入をシナリオ関数自身の仕組みで無力化する。

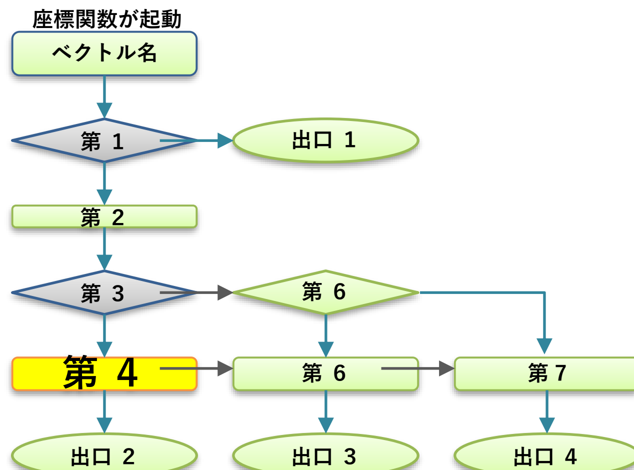
■シナリオ関数でのウイルス無力化とは

1．シナリオ関数はウイルスデータ侵入をR2ベクトル（入力文）のデータ領域である第2規約で受信する。（ベクトルの7つの規約参照）
2．第2規約で受信したウイルスデータは「正常データ」ですので第3規約でデータ内容をチェックします。
　①受信データの属性チェック（あらかじめ特定化された）をします。
　②R2ベクトルの第2規約の実行命令文の存在証明である文脈チェック（変数主語の正統性）をします。
3．第3規約の①，②で正統なデータとして証明されたデータは第4規約に保存されます。
4．R2ベクトルの第4規約で保存されたデータはL4ベクトル，W4ベクトルで変数主語として使用されます。
5．L4ベクトル，W4ベクトルの第3規約で文脈チェックを受け，この繰り返し処理の間にウイルスデータは無力化されます。
6．全ての主語ベクトルは同期アルゴリズムの成立（E41ベクトルON）で実行処理が終わります。（シナリオ関数の実行様相参照）
7．同期関数がE41ベクトルONを確認して，SEPを起動します。
8．SEPは次のシナリオ関数を起動するか，OSに終了宣言し実行権限を戻します。

シナリオ関数は起動してからSEPを呼び出すまでOSの実行権限から切り離されています。

＊ウイルスデータがウイルスプログラムとして起動するのはOSの実行権限の活用が基本になっている

■ベクトルの7つの規約の役割



■ ウイルス対策のポイント

1．ウイルスデータは受信プログラムの受信領域に正常データとして入力されますのでウイルスであれば異常データとして識別する仕組みが必要です
・未知のウイルス対応には正統なデータ生成が必須技術になります

2．正常なデータとして入力されたウイルスデータをウイルスプログラムとして起動させない仕組みが必要です
・データ処理の実行順序権限をOSから切り離すことが必要です

3．正統なデータ生成する仕組みが常に担保され，実行処理が継続できる
・ウイルスデータの無力化とデータ生成は独立して実行される

© 2017 MT　International Co .Ltd.

実践例：シナリオ関数はプログラムの全てのデータ経路を可視化

M T I

従来プログラムのソースコード

主語系譜図作成データ

N	Vector
1	L4_bind_status
2	L4_listen_status
3	L4_client_socket
4	L4_client_socket_close
5	L4_recv_datalength
6	L4_recv_data_filename
7	L4_recv_data_text
8	L4_saveFile_filename
9	W4_saveFile_4
10	W4_send_saveFile_return_4
11	L2_server_socket
12	L2_server_address
13	L2_server_address_len
14	R2_recv_datalengthBuf
15	R2_recv_data
16	L3_listen_status_3
17	L3_client_socket_3
18	L3_client_socket_close_3
19	L3_saveFile_3

9 :W4_saveFile_4

```
FILE* fp = NULL;
if (fopen_s(&fp, (char*)saveFile_filename, "wb") == 0)
{
fputs((const char*)saveFile_text, fp);
fclose(fp);
```

12:L2_server_address

```
memset(&server_address, 0, sizeof(sockaddr_in));//L2_server_address
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
server_address.sin_port = htons(10000);
socklen_t server_address_len = sizeof(sockaddr_in);//L2_server_address_len
```

主語系譜図（全データ経路図）

【始祖】 11：L2_server_socket
【始祖】 13：L2_server_address_len
【始祖】 12：L2_server_address
第1周期
1：L4_bind_status 【最大主語】
16：L3_listen_status_3
第 2 周期
2：L4_listen_status
17：L3_client_socket_3
第 3 周期
3：L4_client_socket 【核主語】
14：R2_recv_datalengthBuf 【終点主語】
第 4 周期
5：L4_recv_datalength
【終点主語】 15：R2_recv_data
第 5 周期
6：L4_recv_data_filename
8：L4_saveFile_filename
7：L4_recv_data_text
19：L3_saveFile_3
第 6 周期
9：W4_saveFile_4 【起点主語】
10：W4_send_saveFile_return_4 【起点主語】
18：L3_client_socket_close_3
第 7 周期
4：L4_client_socket_close
第 8 周期

© 2017 MT　International Co .Ltd.

Ⅱ－２．シナリオ関数化の実践とは

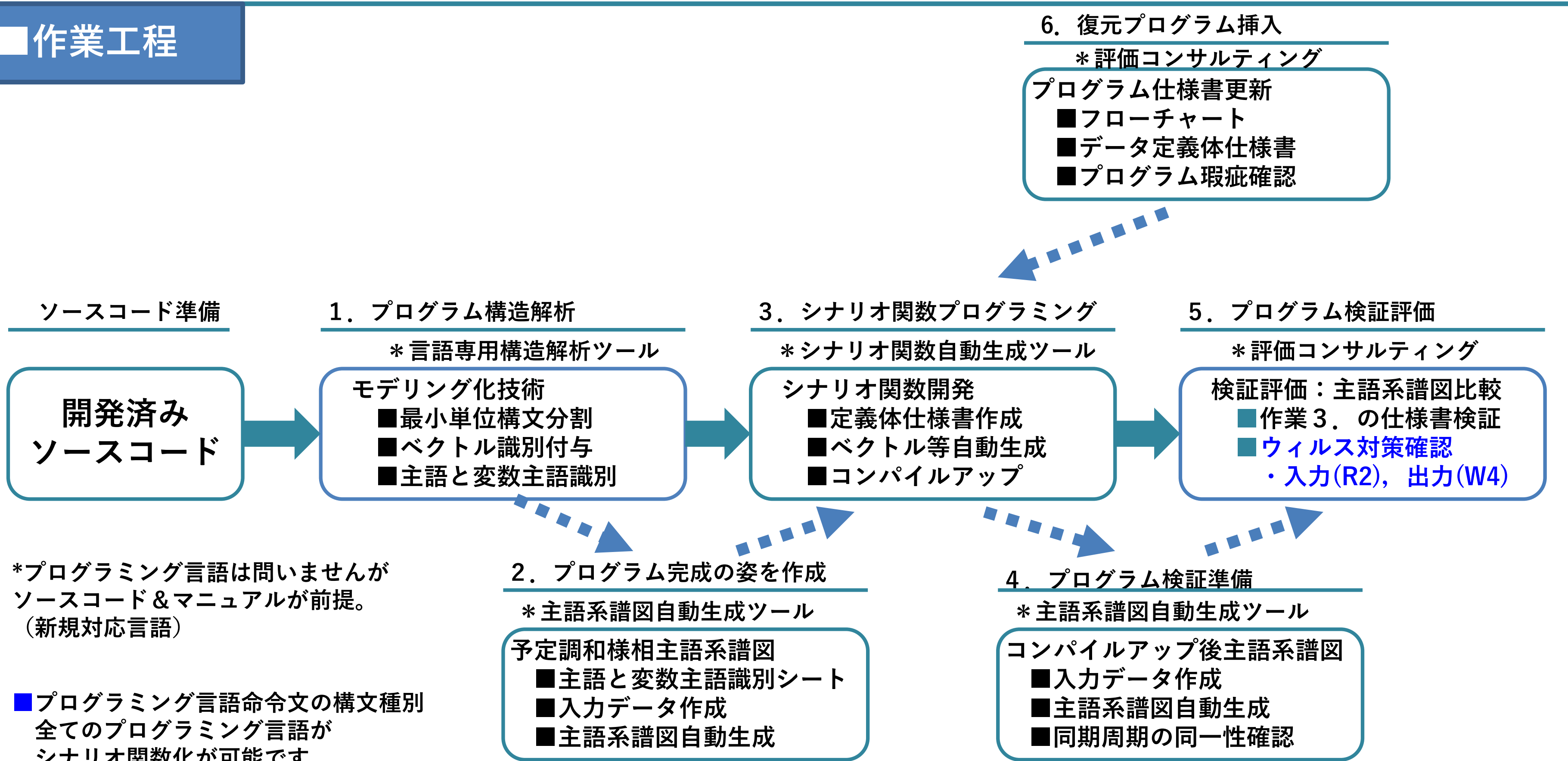
MTI

■　従来プログラムからシナリオ関数化への作業工程

準備１．シナリオ関数化対象DEMOプログラム説明
準備２．対象受信プログラム：操作手順ｖｓソースコード
１．従来プログラムの構造解析（シナリオ関数定義体準備）：構文種別化（１）
１－２．シナリオ関数定義体準備：構文種別化（２）
２．シナリオ関数定義体仕様書
２－２．ベクトルの７つの規約が自動生成を実現
３．DEMOプログラム変数データ
４．DEMOプログラム主語系譜図

© 2017 MTInternational Co,Ltd.

■従来プログラムからシナリオ関数化への作業工程



構文種別：最小単位化（1構文1機能）
1．領域文；領域を定義する叙述
2．関数文；四則演算，論理演算の叙述(代入文，定値・定置文，移動文)
3．入力文；外部情報を取得する為の叙述
4．出力文；外部に情報を送出する為の叙述
5．条件文；真偽値を生成する叙述
6．制御文；実行順序を決める叙述
7．呼出文；部分プログラムを利用する為の叙述
8．翻訳文；コンパイラーが処理する叙述
9．注釈文；

青字はコンパイラー依存部品

© 2017 MTInternational Co,Ltd.

準備１．シナリオ関数化対象DEMOプログラム説明

■クライアントから受信したデータを
　ファイルに保存するプログラム

■ Visual Studio 2015　Community C++

■ファイルの保存先　　　　　　C:￥ｚｚｚ￥

■TCP/ＩＰ

・Listen　IP　　　　Localhost（127.0.0.0）
・Listen　Port　　　10000
・Packet
　Data　Length　4byte
　Data　　　　　0～nbyte

Data Length	Data

Dataは,　ファイル名＋null文字（１byte）+保存するテキスト

■Return　Code

・０ｘ００　正常
・上記以外　エラー

［　備考　］

■クライアントプログラム

・サンプルクライアント（従来法）
・Visual Studio 2015　Community C＃

対象受信プログラムの操作手順（フローチャートの代替）

1. サーバーを起動
   - ・サーバーソケットを初期化
   - ・クライアントソケットを初期化して受信データを待つ

・サーバが受信
   - ・サーバーは受信データの長さを受信
   - ・続けてデータ（ファイル名＋null文字＋内容）を受信
   - ・ファイル名と内容を分離
   - ・C:¥ZZZフォルダに受信ファイル名で内容を保存
   - ・保存に成功したら０、失敗したら１をクライアントに送信
   - ・ソケットをクローズ

2. クライアントを起動

3. クライアントでファイル名とファイルに保存する内容を入力

4. クライアントで「送信」ボタンを押す
   - ・入力データのサイズをサーバーへ送信
   - ・続けて入力データ(ファイル名＋null文字＋内容)を送信
   - ・クライアントはサーバーからの結果の受信を待つ

クライアントが結果を受信
   - ・クライアントの結果欄に「エラー」「正常終了」を表示
   - ・再び、3.へ

© 2017 MTInternational Co,Ltd.

準備１．シナリオ関数化対象DEMOプログラム説明（参考資料）

M T I

対象受信プログラムの操作手順（フローチャートの代替）

1. サーバーを起動

・サーバーソケットを初期化

```
int server_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
int bind_status = bind(server_socket, (struct sockaddr *)&server_address, server_address_len);
int listen_status = listen(server_socket, 10);
```

・ クライアントソケットを初期化して受信データを待つ

```
int client_socket = accept(server_socket, &client_address, &client_address_len);
```

・ サーバが受信

・サーバーは受信データの長さを受信

```
int ret = recv(client_socket, (char*)&datalength, 4, 0);
```

・ 続けてデータ（ファイル名＋null文字＋内容）を受信

```
int retnsize = recv(client_socket, (char*)buf, recvsize, 0);
```

（4バイトずつ読み込み）

・ファイル名と内容を分離

```
strncpy_s(recv_filename, (char *)recv_data, BUF_SIZE);
strncpy_s(saveFile_text, (char *)&recv_data[null_index + 1], BUF_SIZE);
```

・C:¥ZZZフォルダに受信ファイル名で内容を保存

```
strncpy_s(saveFile_filename, "C:¥¥zzz¥¥", BUF_SIZE);

strncat_s(saveFile_filename, (char *)recv_filename, BUF_SIZE -
strlen(saveFile_filename));
if (fopen_s(&fp, (char*)saveFile_filename, "wb") == 0)
{
        fputs((const char*)saveFile_text, fp);
        fclose(fp);
        returnCode = 0;
}
```

・保存に成功したら０、失敗したら１をクライアントに送信

```
send(client_socket, (const char*)&returnCode, 1, 0);
send(client_socket, (const char*)&rlen, 4, 0);
```

・ソケットをクローズ

```
closesocket(client_socket);
```

© 2017 MTInternational Co,Ltd.

準備２．対象受信プログラム：操作手順ｖｓソースコード

ここからいったん　別スライドに切り替わります。　リンク
（準備資料２を印刷して下さい。）

© 2017 MTInternational Co,Ltd.

2．シナリオ関数定義体仕様書（1/2）

主語領域	V記号	ベクトル名	実行命令	変数主語
bind_status	L4	L4_bind_status	bind_status = bind(server_socket, (struct sockaddr *)&server_address, server_address_len);	server_socket,server_address, server_address_len
listen_status	L4	L4_listen_status	listen_status = listen(server_socket, 10);	server_socket,listen_status_3

client_socket	L4	L4_client_socket	if(server_socket,client_socket_3 == 1)
			{
			//デモ用として画面に表示する
			printf("¥nクライアントプログラムから操作してください。¥n¥n");
			struct sockaddr client_address;
			memset(&client_address, 0, sizeof(client_address));
			int client_address_len = sizeof(client_address);
			client_socket = accept(server_socket, &client_address, &client_address_len);
			}
				server_socket,client_socket_3

client_socket_close	L4	L4_client_socket_close	if(client_socket_close_3 == 1)
			{
			closesocket(client_socket);
			}
				client_socket,client_socket_close_3

recv_datalength	L4	L4_recv_datalength	recv_datalength = recv_datalengthBuf;	recv_datalengthBuf

recv_data_filename	L4	L4_recv_data_filename	memset(recv_data_filename, 0, BUF_SIZE);
			strncpy_s(recv_data_filename, BUF_SIZE, (char *)recv_data, BUF_SIZE);
			
				recv_data

recv_data_text	L4	L4_recv_data_text	int null_index = 0;
			for (int i = 0; i < recv_datalength; i++)
			{
			 if (recv_data[i] == 0x00)
			 {
			 null_index = i;
			 break;
			 }
			}
			memset(recv_data_text, 0, BUF_SIZE);
			strncpy_s(recv_data_text, BUF_SIZE, (char *)&recv_data[null_index + 1], BUF_SIZE);
			
				recv_data,recv_datalength

saveFile_filename	L4	L4_saveFile_filename	memset(saveFile_filename, 0, BUF_SIZE);
			strncpy_s(saveFile_filename, BUF_SIZE, "C:¥¥zzz¥¥", BUF_SIZE);
			strncat_s(saveFile_filename, BUF_SIZE, (char *)recv_data_filename, BUF_SIZE - strlen(saveFile_filename));
			
				recv_data_filename

saveFile_4	W4	W4_saveFile_4	int ret = 1;
			FILE* fp = NULL;
			if (fopen_s(&fp, (char*)saveFile_filename, "wb") == 0)
			{
			 fputs((const char*)recv_data_text, fp);
			 fclose(fp);
			 ret = 0;
			}
			//デモ用として画面に表示する
			printf("%s¥n", saveFile_filename);
			printf("%s¥n", recv_data_text);
			
				saveFile_3,saveFile_filename,recv_data_text

send_saveFile_return_4	W4	W4_send_saveFile_return_4	unsigned char ret = (unsigned char)saveFile_4 == 1 ? 0 : 1;
			send(client_socket, (const char*)&ret, 1, 0);
			int rlen = 0;
			send(client_socket, (const char*)&rlen, 4, 0);
			
				client_socket,saveFile_4

server_socket	L2	L2_server_socket	server_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);	

server_address	L2	L2_server_address	memset(&server_address, 0, sizeof(struct sockaddr_in));
			server_address.sin_family = AF_INET;
			server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
			server_address.sin_port = htons(10000);	

server_address_len	L2	L2_server_address_len	server_address_len = sizeof(struct sockaddr_in);	

recv_datalengthBuf	R2	R2_recv_datalengthBuf	unsigned char temp[4];
			int ret = recv(client_socket, (char*)&temp, 4, 0);
			if(ret == 4)
			{
			 recv_datalengthBuf = *(int *)temp;
			}
				client_socket

recv_data	R2	R2_recv_data	memset(recv_data, 0, BUF_SIZE);
			//一度に長いバッファに読み込むのではなく4バイトずつ読み込む
			int rest = recv_datalength;
			int ret = 0;
			while (rest > 0)
			{
			 unsigned char buf[4] = { 0 };
			 int recvsize = rest > 4 ? 4 : rest;
			 int retnsize = recv(client_socket, (char*)buf, recvsize, 0);
			 if (retnsize <= 4)
			 {
			 memcpy_s(&recv_data[ret], BUF_SIZE, buf, retnsize);
			 ret += retnsize;
			 }
			 rest -= recvsize;
			}
			
				recv_datalength,client_socket

listen_status_3	L3	L3_listen_status_3		bind_status
client_socket_3	L3	L3_client_socket_3		listen_status
client_socket_close_3	L3	L3_client_socket_close_3		send_saveFile_return_4
saveFile_3	L3	L3_saveFile_3		saveFile_filename,recv_data_text

© 2017 MTInternational Co,Ltd.

2．シナリオ関数定義体仕様書（2/2）

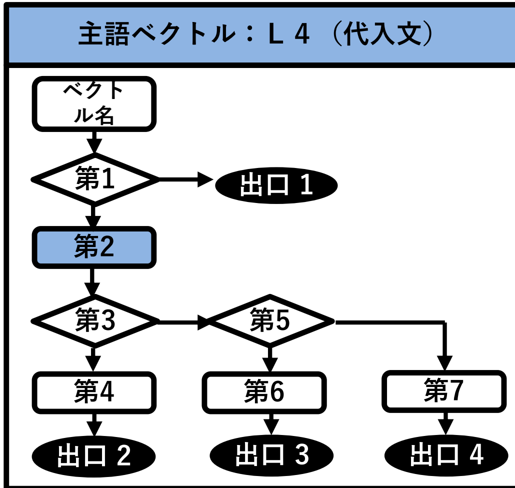
主語領域	V記号	ベクトル名	第4領域名	第4領域コピー領域	型	配列情報	初期化コード	主語がオンであるかのチェックコード	最終成立ベクトル
bind_status	L4	L4_bind_status	bind_status_f4	bind_status_f4_copy	int		bind_status_f4 = -1;	bind_status_f4 != -1	FALSE
listen_status	L4	L4_listen_status	listen_status_f4	listen_status_f4_copy	int		listen_status_f4 = -1;	listen_status_f4 != -1	FALSE
client_socket	L4	L4_client_socket	client_socket_f4	client_socket_f4_copy	int		client_socket_f4 = 0;	client_socket_f4 != 0	FALSE
client_socket_close	L4	L4_client_socket_close	client_socket_close_f4	client_socket_close_f4_copy	int		client_socket_close_f4 = 0;	client_socket_close_f4 != 0	TRUE
recv_datalength	L4	L4_recv_datalength	recv_datalength_f4	recv_datalength_f4_copy	int		recv_datalength_f4 = 0;	recv_datalength_f4 != 0	FALSE
recv_data_filename	L4	L4_recv_data_filename	recv_data_filename_f4	recv_data_filename_f4_copy	char	[BUF_SIZE]	memset(recv_data_filename_f4, 0, BUF_SIZE);	recv_data_filename_f4[0] != 0	FALSE
recv_data_text	L4	L4_recv_data_text	recv_data_text_f4	recv_data_text_f4_copy	char	[BUF_SIZE]	memset(recv_data_text_f4, 0, BUF_SIZE);	recv_data_text_f4[0] != 0	FALSE
saveFile_filename	L4	L4_saveFile_filename	saveFile_filename_f4	saveFile_filename_f4_copy	char	[BUF_SIZE]	memset(saveFile_filename_f4, 0, BUF_SIZE);	saveFile_filename_f4[0] != 0	FALSE
saveFile_4	W4	W4_saveFile_4	saveFile_4_f4	saveFile_4_f4_copy	int		saveFile_4_f4 = 0;	saveFile_4_f4 != 0	FALSE
send_saveFile_return_4	W4	W4_send_saveFile_return_4	send_saveFile_return_4_f4	send_saveFile_return_4_f4_copy	int		send_saveFile_return_4_f4 = 0;	send_saveFile_return_4_f4 != 0	FALSE
server_socket	L2	L2_server_socket	server_socket_f4	server_socket_f4_copy	int		server_socket_f4 = 0;	server_socket_f4 != 0	FALSE
server_address	L2	L2_server_address	server_address_f4	server_address_f4_copy	struct sockaddr_in		memset(&server_address_f4, 0, sizeof(struct sockaddr_in));	server_address_f4.sin_family != 0	FALSE
server_address_len	L2	L2_server_address_len	server_address_len_f4	server_address_len_f4_copy	int		server_address_len_f4 = 0;	server_address_len_f4 != 0	FALSE
recv_datalengthBuf	R2	R2_recv_datalengthBuf	recv_datalengthBuf_f4	recv_datalengthBuf_f4_copy	int		recv_datalengthBuf_f4 = 0;	recv_datalengthBuf_f4 != 0	FALSE
recv_data	R2	R2_recv_data	recv_data_f4	recv_data_f4_copy	char	[BUF_SIZE]	memset(recv_data_f4, 0, BUF_SIZE);	recv_data_f4[0] != 0	FALSE
listen_status_3	L3	L3_listen_status_3	listen_status_3_f4	listen_status_3_f4_copy	int		listen_status_3_f4 = 0;	listen_status_3_f4 != 0	FALSE
client_socket_3	L3	L3_client_socket_3	client_socket_3_f4	client_socket_3_f4_copy	int		client_socket_3_f4 = 0;	client_socket_3_f4 != 0	FALSE
client_socket_close_3	L3	L3_client_socket_close_3	client_socket_close_3_f4	client_socket_close_3_f4_copy	int		client_socket_close_3_f4 = 0;	client_socket_close_3_f4 != 0	FALSE
saveFile_3	L3	L3_saveFile_3	saveFile_3_f4	saveFile_3_f4_copy	int		saveFile_3_f4 = 0;	saveFile_3_f4 != 0	FALSE

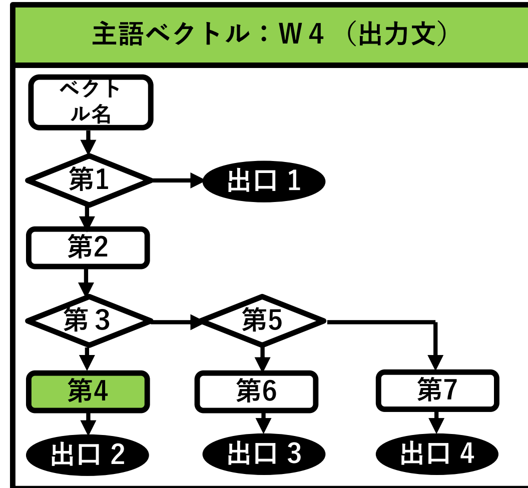
© 2017 MTInternational Co,Ltd.

2－2．ベクトルの７つの規約が自動生成を実現

■公理を前提とするベクトルの７つの規約によって自動生成を実現します
【世界初のプログラミングのロボット生産（自動生産）化を可能にする】

＊主語ベクトル命令文のセット位置：色付け箇所



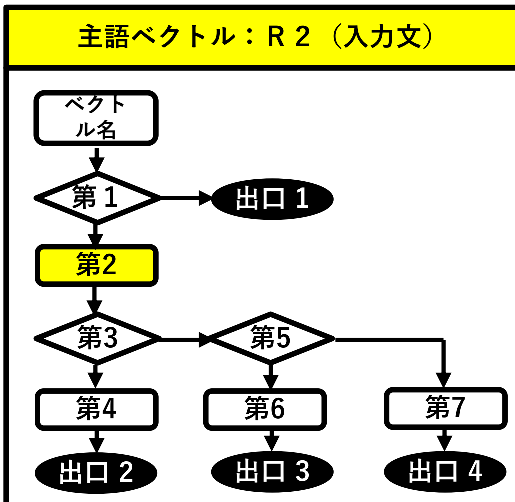


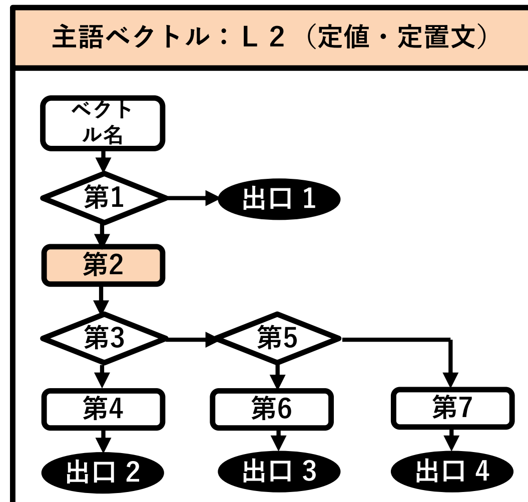
■ベクトルの７つの規約の役割
（最小単位命令文のプログラム定義構造）
第１規約：ベクトルの正統性を判断
第２規約：実行命令文のセット
第３規約：データ成立の正統性を判断
第４規約：成立データの保存領域
第５規約：ベクトルの再起及び停止指示の判断
第６規約：再起指示（同じ座標周期での成立可能性有り）
第７規約：停止指示(成立可能性無し；座標関数移動)

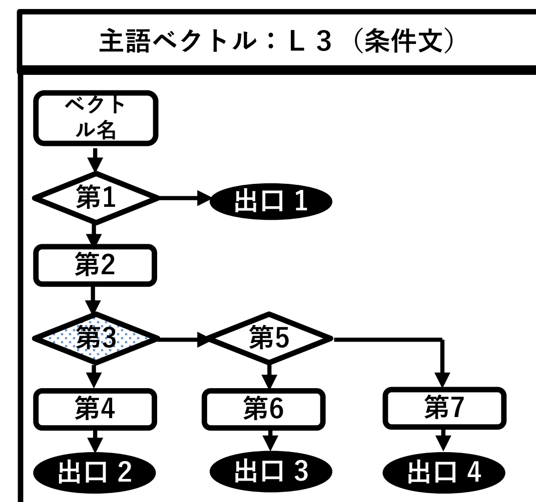
■プログラミング言語命令文の構文種別

コンパイラー非依存構文	コンパイラー依存構文
①代入文：Ｌ４ベクトル	⑧呼出文
②出力文：Ｗ４ベクトル	⑨制御文
③入力文：Ｒ２ベクトル	⑩翻訳文
④定値・定置文：Ｌ２ベクトル	＊１．呼出文，翻訳文は使用方法でL4型制御ベクトルになります
⑤条件文：Ｌ３ベクトル	
⑥領域文	＊２．制御文はシナリオ関数の の仕組みに吸収されます
⑦注釈文	

全てのプログラミング言語がシナリオ関数化が可能です。







© 2017 MTInternational Co,Ltd.

３．DEMOプログラム変数データ（主語系譜図作成の元データ）

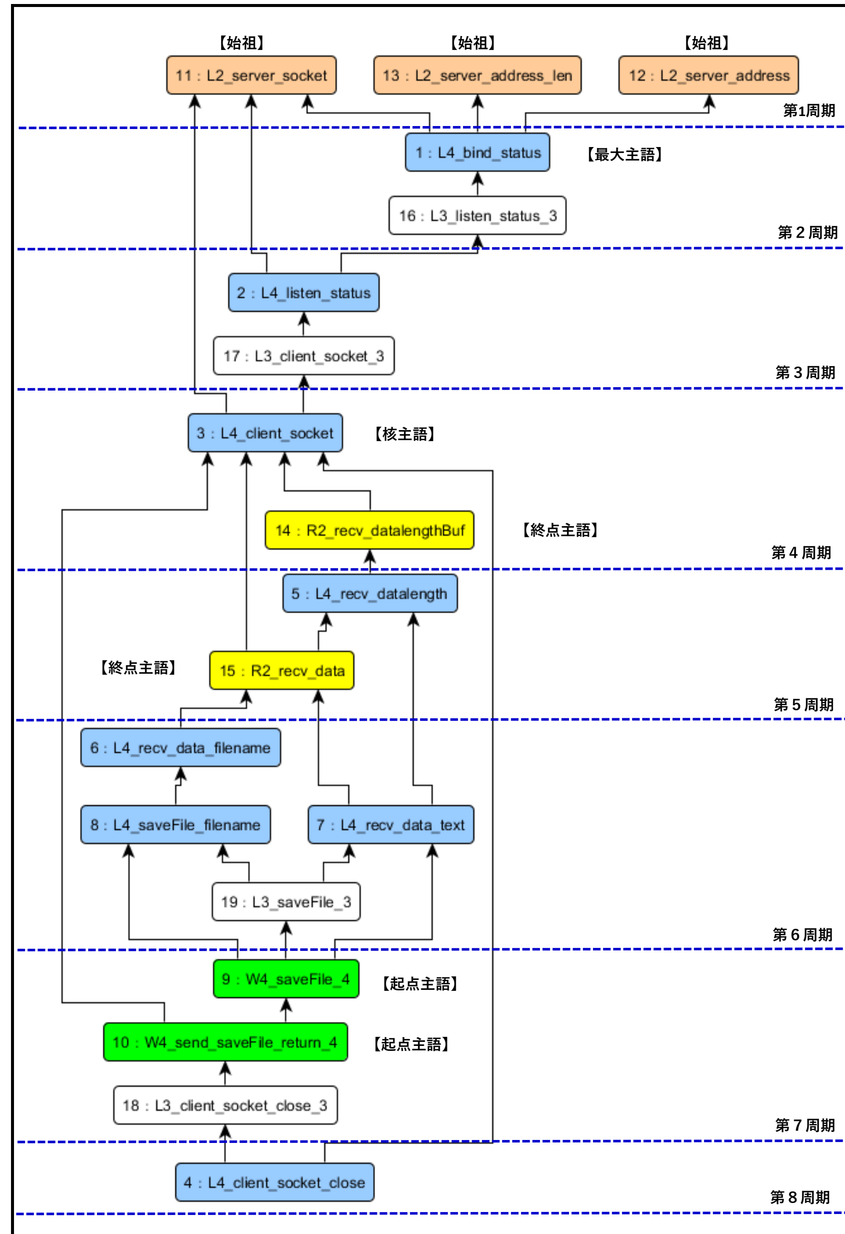
M T I

	主語（生成される変数）	変数主語（生成に関与する変数）		
		1	2	3
1	L4_bind_status	L2_server_socket	L2_server_address	L2_server_address_len
2	L4_listen_status	L2_server_socket	L3_listen_status_3	
3	L4_client_socket	L2_server_socket	L3_client_socket_3	
4	L4_client_socket_close	L4_client_socket	L3_client_socket_close_3	
5	L4_recv_datalength	R2_recv_datalengthBuf		
6	L4_recv_data_filename	R2_recv_data		
7	L4_recv_data_text	R2_recv_data	L4_recv_datalength	
8	L4_saveFile_filename	L4_recv_data_filename		
9	W4_saveFile_4	L4_saveFile_filename	L4_recv_data_text	L3_saveFile_3
10	W4_send_saveFile_return_4	W4_saveFile_4	L4_client_socket	
11	L2_server_socket	NOP		
12	L2_server_address	NOP		
13	L2_server_address_len	NOP		
14	R2_recv_datalengthBuf	L4_client_socket		
15	R2_recv_data	L4_recv_datalength	L4_client_socket	
16	L3_listen_status_3	L4_bind_status		
17	L3_client_socket_3	L4_listen_status		
18	L3_client_socket_close_3	W4_send_saveFile_return_4		
19	L3_saveFile_3	L4_saveFile_filename	L4_recv_data_text	

＊L3_saveFile_3はW4_saveFile_4成立条件（存在証明）として必須なためシナリオ関数化の際にベクトル追加した分です。

© 2017 MTInternational Co,Ltd.

4．DEMOプログラム主語系譜図



© 2017 MTInternational Co,Ltd.

■シナリオ関数で使用される用語集

シナリオ関数で使用される用語に関して

・シナリオ関数の説明で使用されている用語を現状のIT業界用語で理解するのではなく内容でご理解頂ければと思います。
（シナリオ関数での用語は固有名詞です。）

用語	名称の意味及び用語の内容
１．シナリオ関数	発明したデータ結合型プログラムの総称（発明人の名づけ）です。特許申請のプログラム名称になります。
２．ベクトル	ベクトル名称は２種類 ・主語ベクトル：データ生成に関与する命令文が生成するデータ項目名称を主語ベクトル名として使用します。 ・制御ベクトル：シナリオ関数の中で主語ベクトルの実行を制御する目的で作成されていますので制御ベクトルと称します。
３．主語	プログラムが生成するデータ項目名になります。
４．変数主語	生成データ項目の成立に関与するデータ項目名を変数主語と称します。
５．主語系譜図	主語の連鎖構造（生成されたデータ項目のデータ連携図）になります。データフロー図の完全な可視化になります。
６．シナリオ関数の定義式	データ生成に関与する命令文及びデータ項目生成するプログラム群を可視化したのが定義式になります。 ①同期関数とはシナリオ関数の起動及び終了とデータ生成に関与する座標関数を制御するプログラム名称です。 ②座標関数４，２，３は其々が統治する主語ベクトルの実行を制御するプログラム名称です。 ③主語ベクトルは座標関数の起動で第３規約で正統性証明を担保し第４規約のデータ保存領域に保存するプログラムです。 （主語ベクトルプログラムはデータ生成に関与する１構文１機能の命令文を主ベクトル実行領域に定義されます。） ④制御ベクトルのE４１はシナリオ関数の終了宣言するベクトルプログラムです。 ⑤制御ベクトルのE42はシナリオ関数の再帰周期の設定数を超えると座標関数４が緊急停止するベクトルプログラムです。 ⑥制御ベクトルのT４，T2，T31，T32，T33は座標関数の移動を宣言するベクトルプログラムです。
７．シナリオ関数の構造	コンピュータ内部でのシナリオ関数の実行様相を可視化した構造図です。 （シナリオ関数で構成されている各プログラムは終了と同時に次の実行プログラムを呼出す様相を表現しています。）
８．ベクトルの７つの規約	ベクトルプログラムは全て７つの規約に基づいてプログラム化されます。：自動生成（ロボット開発）ができる理由になります。 ・主語ベクトルは命令文が正統なデータ脈絡で成立することを担保しています。：１構文１機能の命令文単位で構成 ・制御ベクトルは其々の機能の実行時期を成立させることを担保しています。 ・全てのプログラミング言語は共通の構文種別を持っていますので主語ベクトル化しますと共通の記述方法になります。
９．従来プログラムの構文種別化	・全てのプログラミング言語は共通の構文種別を持っていますので主語ベクトル化することでシナリオ関数化ができます。 ・データ生成に関与しない領域文，呼出文，翻訳文，注釈文は従来プログラムと同様な記述方法になります。 ・制御文である命令文はシナリオ関数の仕組み及び制御ベクトルに代替えされます：OSの実行権限に依存しない理由です。 ・構文種別化でベクトル名が付与されますのでシナリオ関数自動生成の為の定義体仕様書の準備になります。
１０．主語系譜図作成データ	・主語系譜図（データ連鎖構造）はプログラムが生成する主語及び変数主語データの準備で作成します。 （従来プログラムから同様なデータ抽出で作成は可能ですがデータの正統性は担保されてませんので完全な連鎖構造にはなりません。 シナリオ関数はコンパイルアップで主語系譜図の作成ができますのでテストレスを可能にします。）

© 2017 MTInternational Co,Ltd.