株式会社セック
Systems Engineering Consultants Co., LTD.

RTMSafety™

ホワイトペーパー

第 1 版 2012.5.15

第 2 版 2015.7.10

第 2.2 版 2016.3.8

改訂履歴

Ver.	更新日付	改訂内容	改訂ページ
1.0	2012/05/15	新規作成	－
2.0	2015/09/01	対応プラットフォームの追加。 以下の対応 OS／対応 CPU を追加。 ・ETAS 社製 RTA-OSEK／ZMP 社製 REK-0001 ・OS なし／ZMP 社製 REK-0001 ・OS なし／アルファプロジェクト社製 AP-SH2F-11A ・OS なし／シマフジ電機社製 SEMC2201	P5
2.2	2016/03/08	対応プラットフォームの追記。 以下の対応 OS に対し、対応 CPU を追加。 ・QNX 社製 QNX／ダックス社製 HFBX-6100 ・TOPPERS/ASP／サニー技研社製 SH2A モータ制御ボード	P5

目次

1 RTMSafety 概要

1.1. RTMSafety とは

RTMSafety は、機能安全の国際規格である IEC61508 SIL3 に準拠した、ロボットの安全関連系のためのミドルウェアです。

安全な RT システムを開発するためには、開発プロセス、ハードウェア、およびオペレーティングシステムを含むソフトウェアの開発に対して、IEC61508 に準拠するがあり、多大なコストを要します。RTMSafety は、安全な RT システムのソフトウェアを開発するために必要となる機能を、ライブラリとフレームワークとして提供しています。これらの機能を利用することにより、安全なロボットアプリケーションを効率的に開発することが可能となります。

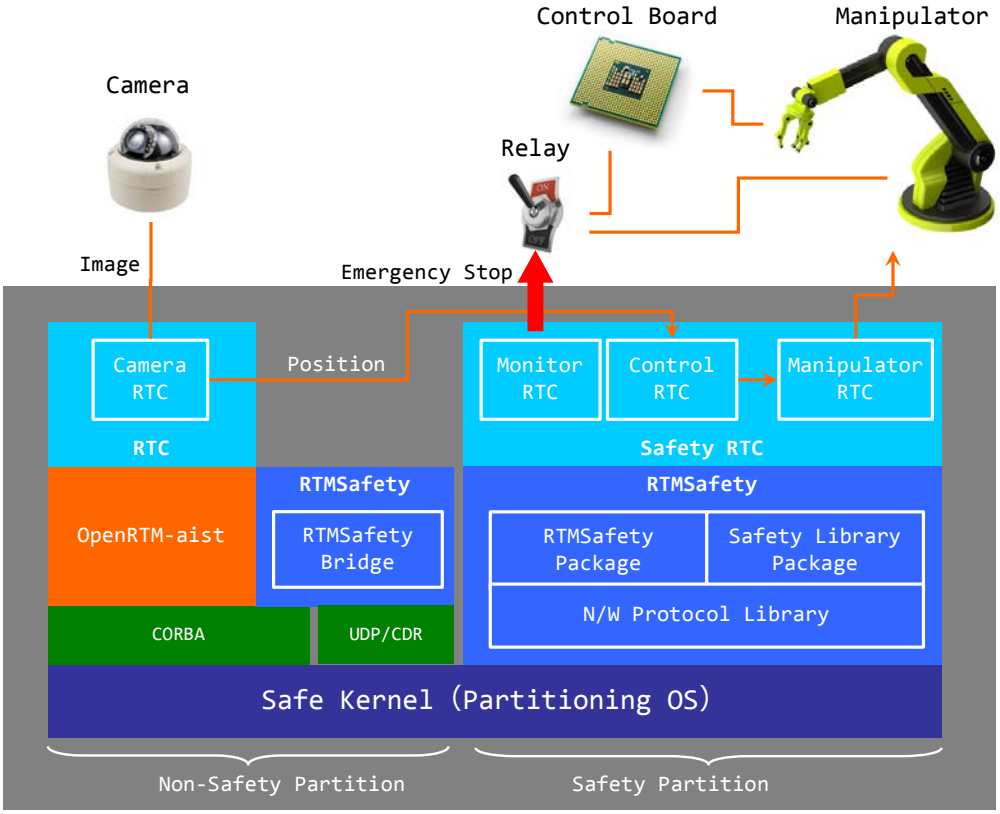

図 1-1　RTMSafety の構成

本製品は下記の 3 つのパッケージと、1 つの関連ツールから構成されています。

- RTMSafety パッケージ
  - コンポーネントフレームワーク（RTMSafety Package）
  - 安全機能ライブラリ（Safety Library Package）
  - ネットワークライブラリ（N/W Protocol Library）
- 関連ツール
  - RTMSafety Bridge

1.1.1. コンポーネントフレームワーク（RTMSafety Package）

コンポーネントフレームワーク（RTMSafety Package）は、RT コンポーネントを開発するための機能を提供する枠組みです。

コンポーネントフレームワークには、状態管理やコンポーネント間の通信、Time Window 制御、エラー処理など、多くのロボットアプリケーションで必要となる定型処理の枠組みが用意されており、ロボットアプリケーション開発者は、これらの枠組みを利用して、効率的にロボットアプリケーションを開発することが可能となります。

また、ロボットアプリケーションをコンポーネント化することにより、システムの保守性、拡張性、再利用性を向上させることが可能となります。

1.1.2. 安全機能ライブラリ（Safety Library Package）

安全機能ライブラリ（Safety Library Package）は、安全関連系のソフトウェアを開発するために必要な機能を提供する枠組みです。

安全機能ライブラリには、コンポーネントの生存監視機能や、自己診断機能を実装するためのフレームワークが用意されています。この枠組みを利用することにより、コンポーネントでエラーが発生した場合や、ハードウェアでエラーが発生した場合に、RT システムを安全に停止させたり、縮退運転に移行させたりするような機能の開発を効率化することが可能となります。

1.1.3. ネットワークライブラリ（N/W Protocol Library）

ネットワークライブラリ（N/W Protocol Library）は、コンポーネント間通信のための機能を提供します。

ネットワークライブラリでは、通信層を抽象化することにより、ネットワークプロトコルの違いを意識することなく、通信機能を利用することが可能となっています。通信層を抽象化することによりコンポーネント間の結合が疎になり、各コンポーネントの保守性向上が期待できます。

本製品では、ネットワークプロトコルとして、カーネルリソース通信と、UDP 通信を提供しています。ただし、UDP 通信機能については、一部 OS 向けのみの対応となります。

1.1.4. RTMSafety Bridge

RTMSafety Bridge は、RTMSafety 上で動作する RT コンポーネント（Safety コンポーネント）と、OpenRTM-aist-1.0 互換コンポーネントの連携を可能とするツールです。OpenRTM-aist は、独立行政法人産業技術総合研究所の開発する OMG RTC Specification に準拠した RT ミドルウェア実装であり、世界中で広く利用されています。

RTMSafety Bridge を利用することにより、Safety コンポーネントの状態を RTSystemEditor などのツールで監視したり、既存の OpenRTM-aist-1.0 互換コンポーネントの資産と連携したりすることが可能となります。

なお、RTMSafety Bridge は、一部 OS 向けのみの対応となります。

1.2. 本製品の安全認証範囲

本製品の安全認証範囲を図 1-2 に示します。

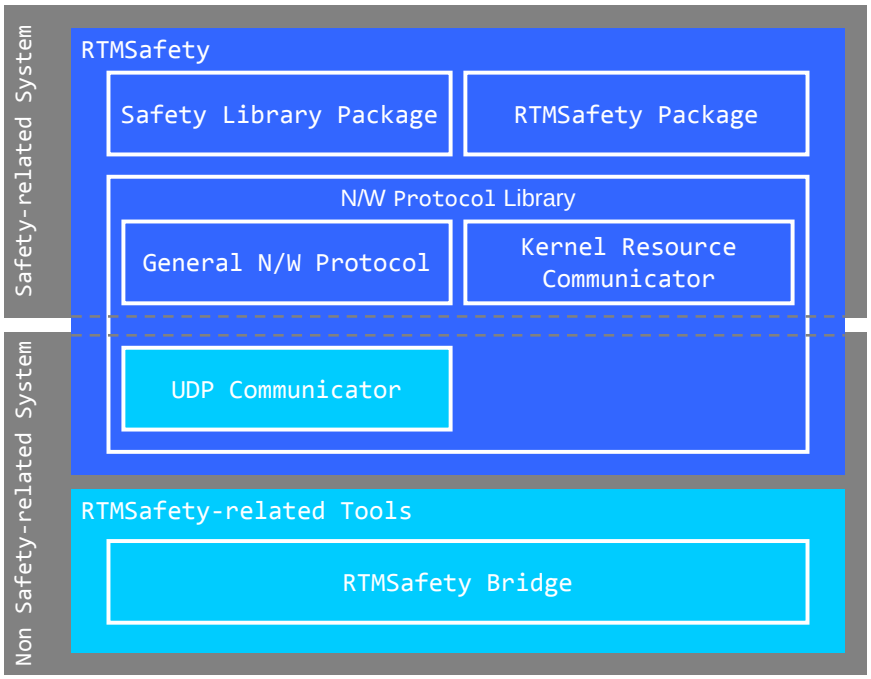

図 1-2　本製品と関連ツールの安全認証範囲

安全認証を取得するには、
すべてのソフトウェアの認証を取得する

コンポーネントフレームワークおよび安全機能ライブラリは、安全関連系の範囲となります。

ネットワークライブラリは、抽象化された通信層（General N/W Protocol）と、カーネルリソース通信機能（Kernel Resource Communicator）については安全関連系となりますが、UDP 通信機能については非安全関連系となります。

また、RTMSafety Bridge は非安全関連系となります。

2 製品仕様

本章では、本製品の対応プラットフォーム、および本製品を構成する各パッケージの構成について説明します。

2.1. 対応プラットフォーム

本製品の対応プラットフォームを表 2-1 に示します。

表 2-1　RTMSafety 対応プラットフォーム

対応 OS	対応 CPU（CPU ボード）
QNX 社製 OS QNX Neutrino RTOS Safe Kernel 1.0	ダックス社製 HFBX-6100 （搭載 CPU インテル社製 i7-610E）
TOPPERS プロジェクト版 OS TOPPERS / ASP 1.3.1	サニー技研社製 SH2A モータ制御ボード （搭載 CPU ルネサス社製 SH72AW SH-2A コア）
ETAS 社製 OS RTA-OSEK	ZMP 社製 REK-0001 （搭載 CPU ルネサス社製 SH72544R SH-2A コア）
OS なし	ZMP 社製 REK-0001 （搭載 CPU ルネサス社製 SH72544R SH-2A コア）
OS なし	アルファプロジェクト社製 AP-SH2F-11A （搭載 CPU ルネサス社製 SH7136 SH-2）
OS なし	シマフジ電機社製 SEMC2201（安全コントローラボード） （搭載 CPU ルネサス社製 V850E2/PG4-L）

※他の OS への移植も承ります。別途ご相談ください。
※他の CPU（CPU ボード）に移植することも可能です。別途ご相談ください。
※価格については、別途お問い合わせください。

2.2. コンポーネントフレームワーク（RTMSafety Package）

2.2.1. RT ミドルウェアと RT コンポーネント

RT ミドルウェアは、ロボット機能要素（以降、「RT 機能要素」と称する）のソフトウェアモジュールを複数組み合わせてロボットシステムを構築するためのソフトウェアプラットフォームです。RT 機能要素をソフトウェアモジュール化したものを RT コンポーネントと呼びます。RT コンポーネントの仕様は OMG において国際標準化されており、OMG の RTC Specification に準拠した RT ミドルウェア実装の 1 つとして OpenRTM-aist があります。

RTMSafety は、OMG の RTC Specification のサブセットに準拠した RT ミドルウェア実装です。

安全関連系のソフトウェア開発において、動的コンフィギュレーションの利用は制限されているため、本製品では RTC Specification に含まれるコンフィギュレーション等の機能は提供せず、その利用を制限しています。また、動的なメモリ確保の利用も制限されているため、可変長配列の利用や、動的なインスタンスの生成機能は提供していません。

一方、安全なソフトウェアを開発するために必要となる、Time Window 機能、エラー通知機能、自己診断機能など、RTC Specification では規定されていない独自の機能を提供しています。

RTMSafety が提供するコンポーネントフレームワークの構成を図 2-1 に示します。

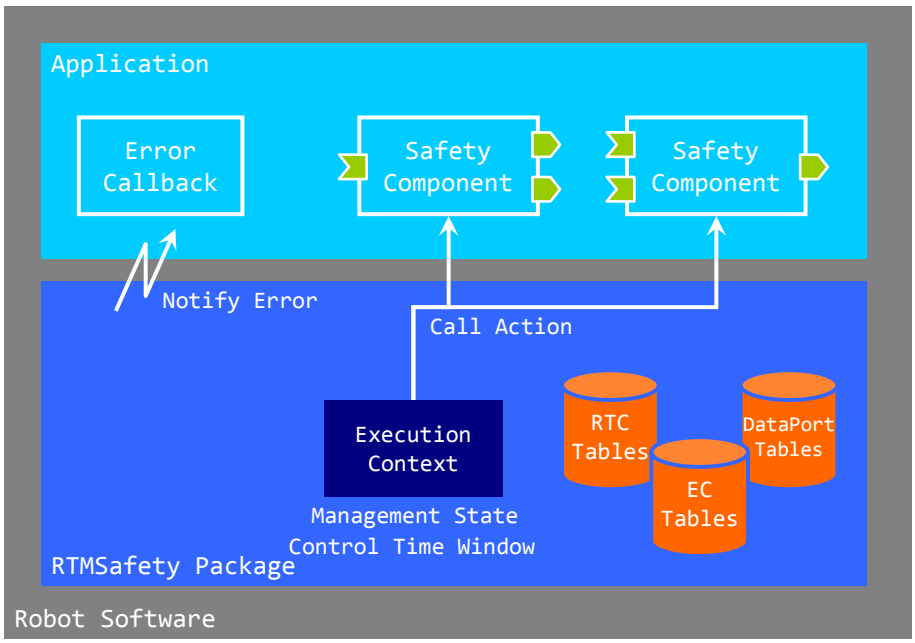

図 2-1　コンポーネントフレームワーク

アプリケーション開発者は、コンポーネントフレームワークの提供するライブラリおよびフレームワークを利用し、RT コンポーネント、およびエラー発生時のコールバックを実装します。

RT コンポーネントは、実行コンテキストにより管理されており、状態に応じたコンポーネントアクションの呼び出し、Time Window に応じた実行タイミングの制御が行われます。

また、データポート機能を利用することにより、RT コンポーネント間においてデータ通信を行うことが可能となります。

アプリケーションや RTMSafety においてエラーが発生した際には、コールバック関数によりアプリケーションに通知が行われます。アプリケーション開発者は、エラーコールバック関数の中で、エラーに応じた復旧処理を実装します。

各 RT コンポーネントの構成、データポートの構成、実行コンテキストの構成については、各種テーブルにより管理しています。

2.2.2. RT コンポーネントライフサイクル管理

RT コンポーネントは、OMG の RTC Specification に準拠した状態遷移に従って動作します。ただし、本製品では、コンポーネントの安全性を考慮し、アプリケーション内での明示的な start、stop、activate_component、deactivate_component の呼び出しを禁止しています。

RT コンポーネントライフサイクル管理機能における状態遷移を図 2-2 に、状態一覧を表 2-2 に、コンポーネントアクション一覧を表 2-3 に示します。

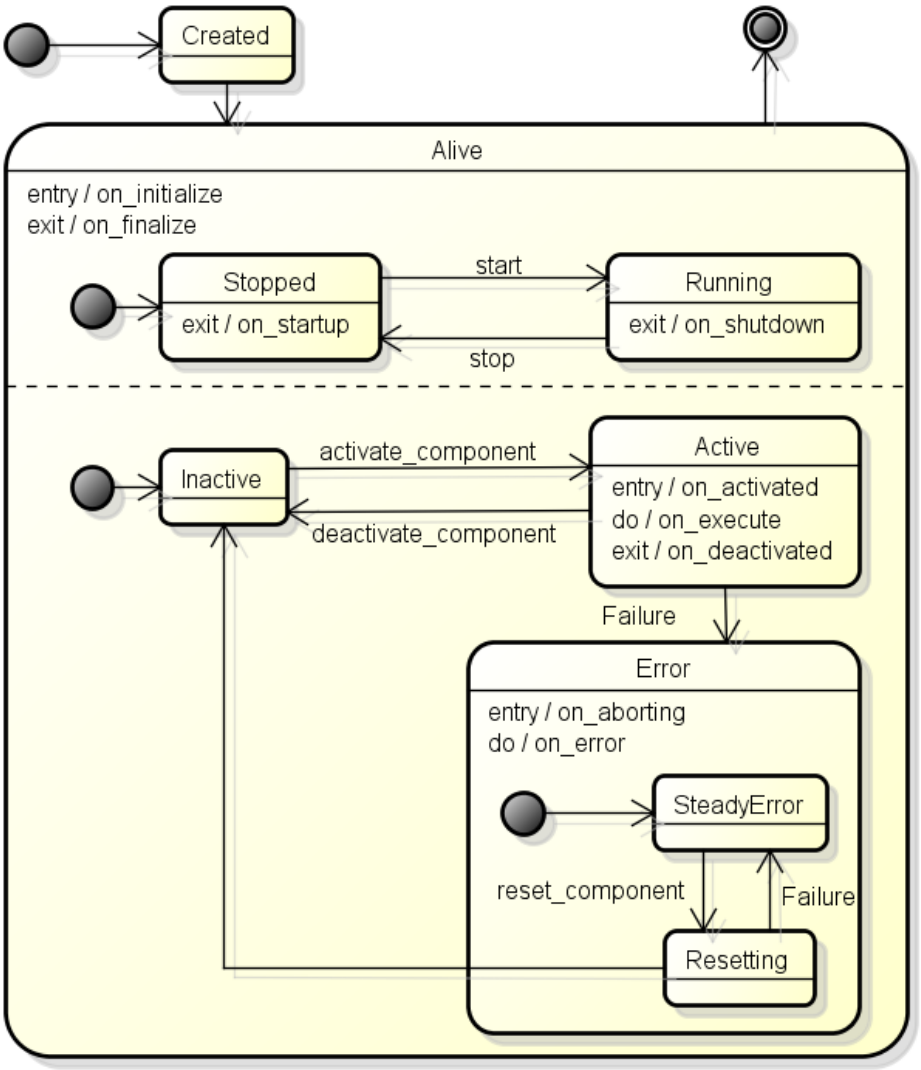

図 2-2　RT コンポーネントライフサイクル

表 2-2　RT コンポーネントライフサイクルにおける状態一覧

状態	説明
Created	RT コンポーネントのインスタンスが生成された状態です。 自動的に Alive 状態に遷移します。
Alive	RT コンポーネントが生存している状態です。
Stopped	実行コンテキストが停止している状態です。 本製品ではサポートしません。
Running	実行コンテキストが起動している状態です。 本製品では実行コンテキストは常に Running 状態です。
Inactive	RT コンポーネントが非活性状態です。 データポートの接続確認がすべて完了すると、自動的に Active 状態に遷移します。
Active	RT コンポーネントが活性状態です。 RT コンポーネントがこの状態にある場合、メインロジック（on_execute）を周期的に実行します。 メインロジックでエラーが発生した場合、Error 状態に遷移します。
Error	RT コンポーネントがエラー状態です。
SteadyError	エラーからの復旧待ち状態です。 RT コンポーネントがこの状態にある場合、エラー処理（on_error）を周期的に実行します。 reset_component イベントの発生により、Resetting 状態に遷移します。
Resetting	RT コンポーネントのリセットを実行する状態です。 RT コンポーネントのリセットに成功した場合は Inactive 状態に遷移し、失敗した場合は SteadyError 状態に遷移します。

表 2-3　コンポーネントアクション一覧

アクション名	概要
on_initialize	RT コンポーネントが Alive 状態に遷移した際に 1 度だけ呼ばれます。
on_activated	RT コンポーネントが Active 状態に遷移した際に 1 回呼ばれます。
on_deactivated	RT コンポーネントが Active 状態から Inactive 状態に遷移した際に 1 回呼ばれます。
on_execute	RT コンポーネントが Active 状態にある場合に周期的に呼ばれます。
on_aborting	RT コンポーネントが Active 状態から Error 状態に移行する際に 1 回呼ばれます。
on_error	RT コンポーネントが Error 状態にある場合周期的に呼ばれます。
on_reset	RT コンポーネントが Error 状態から復帰する際に 1 回呼ばれます。
on_startup	実行コンテキストが開始する際に 1 回呼ばれます。
on_shutdown	実行コンテキストが停止する際に 1 回呼ばれます。 本製品ではサポートしません。
on_finalize	RT コンポーネントを終了する際に 1 度だけ呼ばれます。

本製品では RT コンポーネントの動的な生成を禁止しています。
RTC 生成テーブルに RT コンポーネントの情報を記述することにより、システムの起動時に RT コンポーネントが自動的に生成されます。

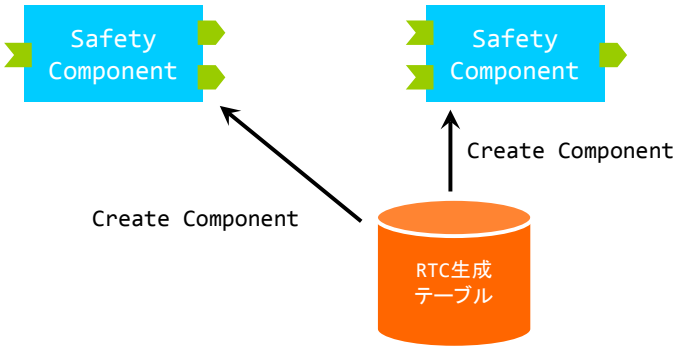

図 2-3　RT Object 生成機能動作概要

RT コンポーネント生成テーブルには以下を設定する必要があります。

- RT コンポーネントのインスタンス領域（RTC 管理テーブル）
- RT コンポーネントの生成関数
- RT コンポーネントの ObjectKey（ObjectKey については 2.4.3 参照）
- RT コンポーネントが持つデータポートの ObjectKey

2.2.3. Time Window

RT システムにおいて機能安全を実現するためには、ソフトウェアのリアルタイム性が必要となります。すなわち、周期処理の最大実行時間が予測可能でなければなりません。

RTMSafety では、各 RT コンポーネントのリアルタイム性を実現するため、タスクの実行タイミングを制御するフレームワークを提供しています。

2.2.3.1 Major Time Window と Minor Time Window

RTMSafety におけるタスクの動作方針を図 2-4 に示します。RTMSafety では、すべてのタスクは基本的に一定の周期で繰り返し実行されます(初期化タスクおよびUDP受信タスクを除く)。このとき、周期実行の最小の単位となる時間幅を Major Time Window と呼び、タスクは必ずこの Major Time Window の整数倍の周期で動作することとなります。

Major Time Window は、さらに小さな時間幅の Minor Time Window で構成されます。タスク毎に Minor Time Window の消費数を割り当てます。このとき、1 つの Minor Time Window に、複数のタスクを割り当てることはできません。

タスクの実際の実行時間が、割り当てられた Minor Time Window よりも大きくなった場合、アプリケーションにコールバック関数によりエラーとして通知されます。

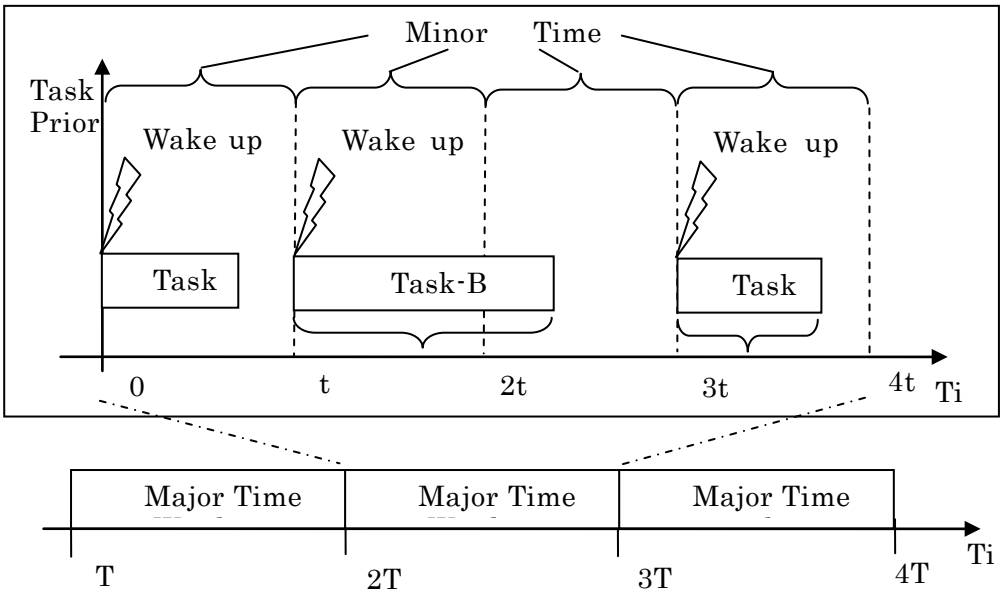

図 2-4　タスク動作方針

2.2.3.2 タスク構成

RTMSafetyでは、表 2-4に示すタスクを有しています。アプリケーション開発者は、これらのタスク毎にTime Windowによるスケジュールを設定する必要があます。

なお、初期化タスクとUDP受信タスク以外のタスクは、複数のタスクが並列に動作するように設定することはできません。

表 2-4 タスク一覧

タスク名	タスク処理内容	優先度
実行コンテキスト	RTコンポーネントの実行を管理するタスク。 実行コンテキスト生成テーブルの設定により、複数個定義することが可能です。	中
データポート接続確認タスク	データポート間の接続確認を行うタスク。	中
生存状況監視タスク	RTコンポーネントの生存状況を監視するタスク。	中
生存情報収集タスク	RTコンポーネントの生存情報を収集するタスク。	中
General N/Wタスク	メッセージの送受信を行うタスク。	中
UDP受信タスク	UDPメッセージを受信するタスク。 QNX向けRTMSafetyにのみ存在します。	低
初期化タスク	各タスクの初期化を行います。	低

2.2.3.3 実行コンテキスト

RT コンポーネントの状態管理と、実行タイミングの制御を行うためのタスクとして、実行コンテキストがあります。

実行コンテキストは、必要に応じて複数作成することができ、1 つの実行コンテキストは複数の RT コンポーネントを有することができます。

RTMSafety ではタスク毎に実行周期を指定でき、実行周期が同じコンポーネントのグループ毎に実行コンテキストを用意することになります。

例えば、図 2-5 に示すように、高周期で動作させたいコンポーネント群と、低周期で動作させたいコンポーネント群を別々の実行コンテキストに所属させるという使い方が可能です。

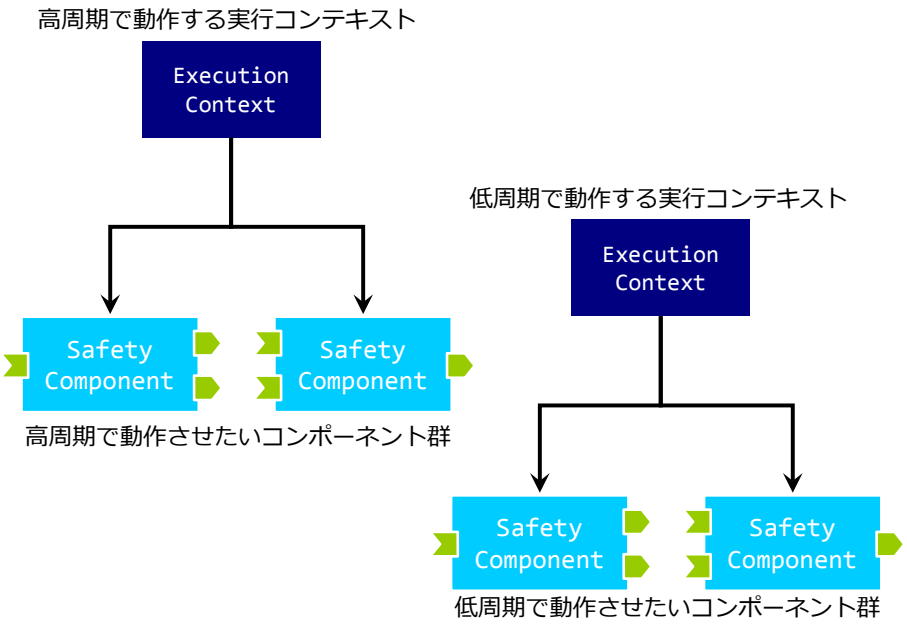

図 2-5　周期によるコンポーネントのグループ化

2.2.3.4 タスクと実行コンテキストの生成

RTMSafety では、実行コンテキストの動的な生成を禁止しています。
タスク生成テーブルおよび実行コンテキスト生成テーブルに、タスクと実行コンテキストの情報を記述しておくと、システムの起動時に実行コンテキストが自動的に生成されます。

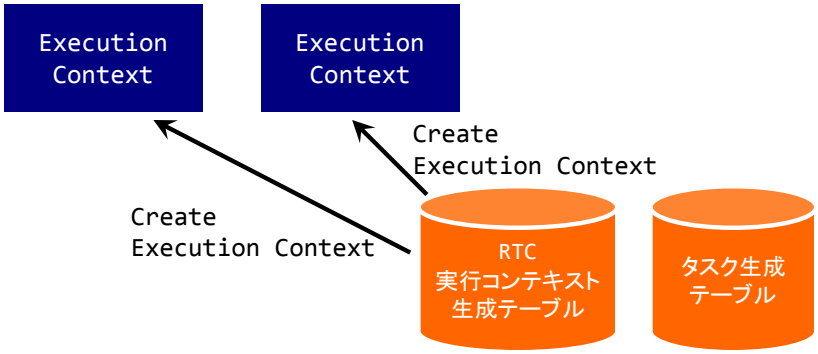

図 2-6　実行コンテキスト生成機能動作概要

タスク生成テーブルには以下を設定する必要があります。

- タスクのインスタンス領域（タスク管理テーブル）
- タスクの ObjectKey
- タスク優先度

また、実行コンテキスト生成テーブルには以下を設定する必要があります。

- 実行コンテキストのインスタンス領域（実行コンテキスト管理テーブル）
- 実行コンテキストのインスタンス領域（実行コンテキスト管理テーブル）
- タスクの ObjectKey
- 管理対象である RT コンポーネントの ObjectKey

2.2.4. データポート機能

データポートは、RT コンポーネント間でデータのやりとりを行うための機能です。データポートの構成を図 2-7 に示します。

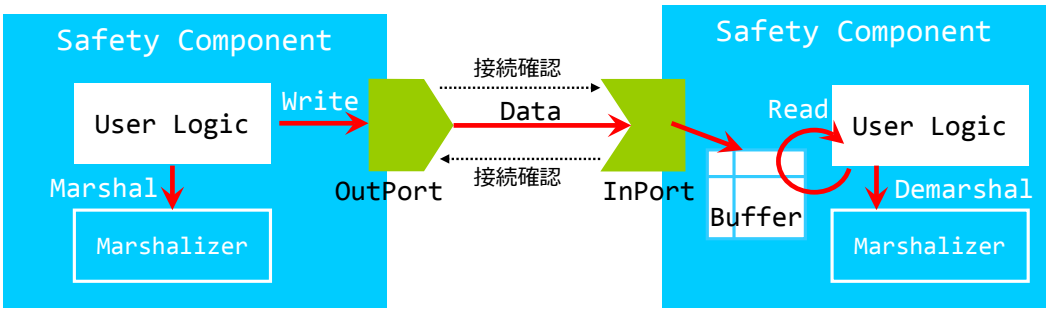

図 2-7　データポート機能動作概要

データポートは、データを出力するための機能である OutPort と、データを入力するための機能である InPort から構成されます。

出力側のコンポーネントのユーザロジックにおいて OutPort にデータを書き込むと、接続された InPort にデータが送信され、そのデータは一旦ダブルバッファに蓄えられます。入力側のコンポーネントのユーザロジックにおいて、バッファからデータを読み込みます。

データポート間において送受信するデータのフォーマットは CDR 形式です。Marshalizer を利用して、ユーザロジックからマーシャリング、デマーシャリングを行う必要があります。マーシャリング・デマーシャリングについては、2.4.2 を参照してください。

また、データポート間の接続確認を行うため、OutPort と InPort の間では相互に定期的な接続確認メッセージを送受信しています。この接続確認メッセージが一定時間受信できない状態が続くと、データポートの接続が切断したと見なし、アプリケーションに対してコールバック関数による通知を行います。

なお、一旦データポートが切断した後、再び接続確認メッセージをお互いに受信することができるようになれば、接続状態は復旧します。

RTMSafety では、データポートの動的な生成と、データポート間の動的な接続および切断処理を禁止しています。

タスク生成テーブルおよび実行コンテキスト生成テーブルに、タスクと実行コンテキストの情報を記述しておくと、システムの起動時に実行コンテキストが自動的に生成されます。

データポート生成テーブルにデータポートの情報を記述しておくと、システムの起動時にデータポートが自動的に生成されます。

また､データポート接続管理テーブルにデータポートの接続情報を記述しておくと、RT コンポーネントの初期化時にデータポートが自動的に接続されます。

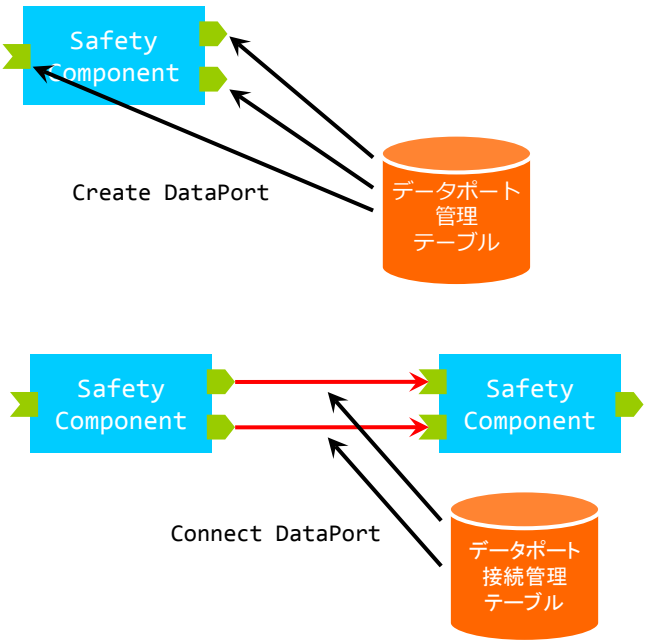

図 2-8　データポート生成機能、データポート接続管理機能動作概要

データポート生成テーブルには以下を設定する必要があります。

- データポートのインスタンス領域（データポート管理テーブル）
- データポートの向き
- データポートが送受信するデータのサイズ(byte 単位)
- データポートの ObjectKey

データポート接続管理テーブルには、以下を設定する必要があります。

- 接続する InPort と OutPort の ObjectKey
- 送信するデータサイズ

2.2.5. エラー通知機能

RTMSafety では、RTMSafety 内もしくは RT コンポーネントにおいて何らかのエラーを検出した場合、コールバック関数により、アプリケーションに通知を行います。

RT システムを開発する際には、故障モードの分析を行い、各故障への対策としてシステムの安全な停止や、縮退運転モードへの切り替えなどを行います。これらの処理の実装をエラーコールバックに実装する必要があります。

表 2-5　エラーコールバック一覧

コールバック名	コールバック発生条件
RtmMnt_onDisconnection	データポートの接続確認、データ送信が行えなかった場合に呼び出されます。
RtmMnt_onRtcLifeCycleError	RT コンポーネントが Error 状態になった場合に一度だけ呼び出されます。
RtmMnt_onRtcLifeCycleTimeOut	RT コンポーネントの生存情報が一定時間確認できなかった場合に呼び出されます。
RtmMnt_onAssertionError	RTMSafety 内部で Assertion エラーが発生した場合に呼び出されます。
RtmMnt_onReceiveDataError	受信処理において不正なデータを受信した場合に呼び出されます。
RtmMnt_onSendDataError	データ送信に失敗した場合に呼び出されます。
RtmMnt_onNetworkError	ネットワーク関連の処理でエラーが発生した場合に呼び出されます。
RtmMnt_onFailKernelOperation	システムコールの呼び出しでエラーが発生した場合に呼び出されます。
RtmMnt_onNotFoundObjectKey	不正な ObjectKey を検出した場合に呼び出されます。
RtmMnt_onTaskOverRun	タスクの処理が指定した時間内に完了しなかった場合に呼び出されます。
RtmMnt_onFailInitializeTask	初期化処理に失敗した場合に呼び出されます。

2.2.6. RAS 機能

RTMSafety では、RT システムの保守性を向上させるための機能として、下記のような RAS 機能を提供しています。

- ロギング機能
- Assertion マクロ
- 割り込み禁止 / 割り込み解除

ロギング機能は、エラーログおよび実行ログを蓄積するための機能です。
RTMSafety 内で発生したエラー情報をメモリ上に蓄積しておき、デバッガやツールを利用してログ内容をダンプし、エラー原因の解析を行うことができます。

Assertion マクロは、任意の変数が指定の条件に合致するか否かのチェックを行うための機能です。Assertion マクロは、RTMSafety 内の関数呼び出しにおいて、変数の内容が指定の条件に合致しない場合、エラー通知機能のコールバック関数により、アプリケーションにエラー通知を行います。

割り込み禁止 / 割り込み解除機能は、OS 毎に異なる割り込み禁止 / 割り込み解除命令の API をラップして、共通的な API としたものです。
割り込み禁止 / 割り込み解除機能を利用すると実行コンテキストの動作が変わることがあるため、注意して使用してください。また、割り込み禁止時間は可能な限り短くしてください。

2.2.7. インスタンス数とデータサイズの上限

本製品では、RT コンポーネントやデータポートのインスタンス数、データサイズの範囲を表 2-6 に示すように定義しています。

表 2-6　インスタンス数とデータサイズの上限

項目	QNX 向け RTMSafety	TOPPERS 向け RTMSafety
RT コンポーネント数	1～16	1～16
1 つのコンポーネントのデータポートの数	0～8	0～8
システム内でのデータポート数の上限	0～64	0～16
データポートで利用可能な最大データサイズ	最大 512Byte	最大 64Byte
OutPort と InPort の接続数	1 つの OutPort に対して最大 4 つの InPort の接続	

2.2.8. OpenRTM-aist との機能比較

本製品と OpenRTM-aist-1.0 の機能比較を表 2-7 に示します。

表 2-7　RTMSafety と OpenRTM-aist の機能比較

項目	OpenRTM-aist-1.0	RTMSafety
コンポーネント		
コンポーネントタイプ	DataFlow Component	DataFlow Component
コンポーネントアクション	on_initialize on_activated on_deactivated on_execute on_aborting on_error on_reset on_finalize on_startup on_shutdown on_rate_changed on_state_update	on_initialize on_activated on_deactivated on_execute on_aborting on_error on_reset on_finalize on_startup
データポート		
ポートの接続	動的に接続切断が可能	接続は静的に指定する。 動的な接続/切断は不可
データフロータイプ	push, pull	push
サブスクリプションタイプ	Flush, New, Periodic	New
バッファリング方式	InPort：リングバッファ OutPort：なし （独自バッファに差し替え可能）	InPort：ダブルバッファ OutPort：なし
データフォーマット	CDR 形式	CDR 形式（一部未対応）
独自データ型	利用可能：マーシャリング処理は IDL コンパイラにより自動生成	利用可能：マーシャリング処理はアプリケーション開発者が実装
実行コンテキスト		
実行コンテキストの動作	デフォルトの PeriodicExecutionContext は、一定周期で動作。 （独自実行コンテキストに差し替え可能）	Time Windowにより厳密に定義する。
コンポーネントの操作	プロセス外から下記の操作を呼び出し可能。 activate_component, deactivate_component, reset_component	プロセス外からの呼び出し不可。 同一プロセスからは reset_component のみ利用可能。 activate_component, deactivate_componentは明示的に呼び出すことを許可しない。

項目		OpenRTM-aist-1.0	RTMSafety
	動作周期の動的な変更	対応	未対応
	実行コンテキストの拡張	対応	未対応
サービスポート		対応	未対応
SDO			
	コンフィギュレーション	対応	未対応
	複合コンポーネント	対応	未対応

2.3. 安全機能ライブラリ（Safety Library Package）

2.3.1. 安全機能ライブラリの構成

安全機能ライブラリ（Safety Library Package）は、安全関連系のソフトウェアを開発するために必要な機能を提供する枠組みです。
安全機能ライブラリは、コンポーネントの生存監視機能と、自己診断機能を実装するためのフレームワークから構成されます。

2.3.2. 生存監視機能

生存監視機能は、一定周期で起動し RT コンポーネントの状態の収集と、監視対象の RT コンポーネントの生存状況の監視を行う機能です。
監視対象の RT コンポーネントが Error 状態の場合は、2.2.5 に示すエラー通知機能を利用し、アプリケーションに対してコールバック通知を行います。
なお、監視対象とする RT コンポーネントは生存情報監視管理テーブルに設定します。

2.3.3. 自己診断機能

自己診断機能は、ソフトウェアの暴走を検知するための WDT（ウォッチドッグタイマー）機能と、ハードウェア要素の故障検出のための診断処理を実行するためのフレームワークである診断処理実行機能から構成されます。

WDT 機能は、一定時間ソフトウェアからの応答がない場合、ハードウェアリセットを行ったり、ソフトウェアを再起動したりするための機能です。RTMSafety が提供する WDT 機能は、各 OS が提供する WDT 機能をラップしたものであり、OS 毎に機能が異なります。

診断処理実行機能は、アプリケーション開発者が実装した診断処理を実行し、その結果何らかの故障を検出した場合、アプリケーション開発者がコールバック関数を呼び出す仕組みを提供します。診断処理およびハードウェア故障を検出した際のコールバック関数は自己診断処理管理テーブルに設定します。

2.4. ネットワークライブラリ（N/W Protocol Library）

2.4.1. ネットワークライブラリの構成

ネットワークライブラリ（N/W Protocol Library）は、コンポーネント間通信のための機能を提供するライブラリです。

ネットワークライブラリの構成を図 2-9 に示します。

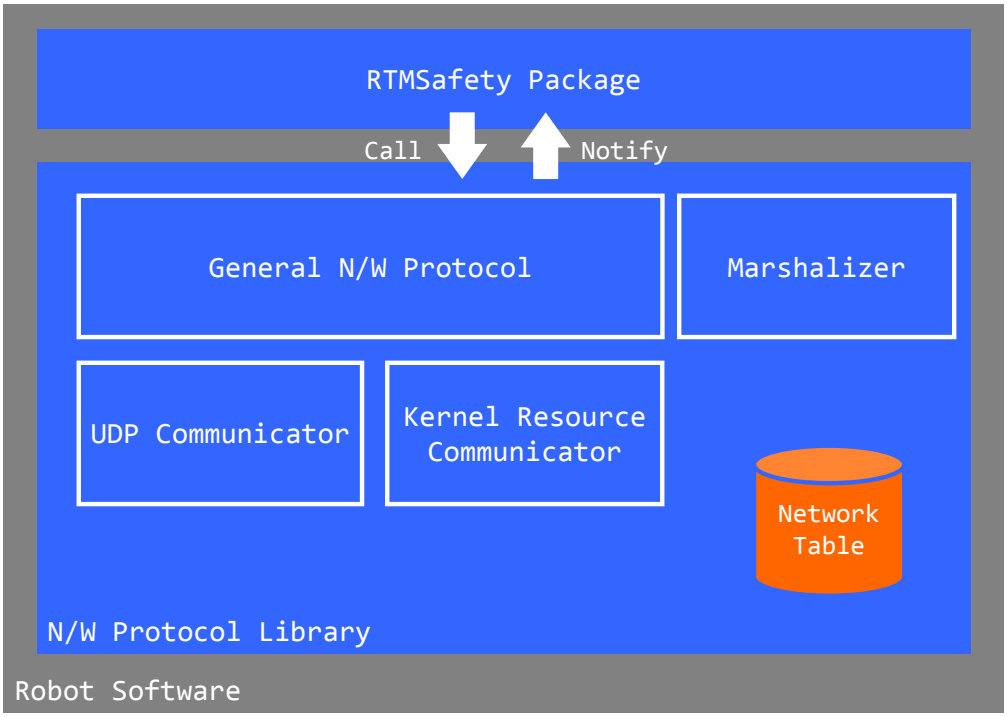

図 2-9　ネットワークライブラリの構成

Kernel Resource Communicator は、メッセージキューなどのカーネルリソース通信を利用した通信を実現する機能です。

UDP Communicator は、UDP 通信を利用した通信を実現する機能です。

General N/W Protocol は、ネットワークプロトコルに依存しないメッセージフォーマットに従った通信を行うレイヤーです。Kernel Resource Communicator と UDP Communicator は抽象化されており、アプリケーションの開発者はどちらの通信方式を利用する場合でも、プログラム上は意識する必要がありません。(利用プロトコルはテーブルに記述します)

Marshalizer は、CDR 形式のデータフォーマットに従って、メッセージの変換を行うための機能です。

2.4.2. データフォーマット

RTMSafety では、コンポーネント間のメッセージ交換を行う際のメッセージフォーマットとして、OMG の GIOP で規定されている CDR（Common Data Representation）を採用しています。

ただし、ソフトウェアの安全性を考慮し、サイズが動的に変化するような型（String, Sequence, Any）や、利用頻度の少ない型については、その利用を禁止しています。（詳細については表 2-8 を参照してください）

ネットワークライブラリでは、RT コンポーネントが扱う構造体をバイナリ配列に変換する処理（マーシャリング）と、受信したバイナリデータを構造体に変換する処理（デマーシャリング）を行うライブラリとして、Marshalizer を提供しています。

Marshalizer では、CPU 毎のバイトオーダの違いを考慮してマーシャリング/デマーシャリングを行っているため、アプリケーション開発者はバイトオーダの違いを考慮する必要はありません。

なお、本製品では IDL コンパイラを提供していないため、マーシャリング/デマーシャリングの処理は、アプリケーション開発者が手動で実装する必要があります。

また、メッセージフォーマットとして CDR 形式を採用していることにより、2.5 に示す RTMSafety Bridge を利用することで、OpenRTM-aist-1.0 互換コンポーネントとの、シームレスなデータ交換が可能となっています。

表 2-8 に、本製品が対応する CDR のデータ形式を示します。

表 2-8 CDR 対応データ形式一覧

型名	説明	RTMSafety での対応
Primitive Types		
char	1Byte 文字	対応
wchar	ワイド文字	未対応
octet	符号無し 1Byte 整数	対応
short	符号有り 2Byte 整数	対応
unsigned short	符号無し 2Byte 整数	対応
long	符号有り 4Byte 整数	対応
unsigned long	符号無し 4Byte 整数	対応
long long	符号有り 8Byte 整数	未対応
unsigned long long	符号無し 8Byte 整数	未対応
float	単精度浮動小数点	対応
double	倍精度浮動小数点	対応
long double	拡張倍精度浮動小数点	未対応
boolean	論理型	対応
enum	列挙体	short 型で代用することで対応可能
Constructed Types		
Struct	構造体	対応
Union	共用体	未対応
Array	固定長配列	対応
Sequence	可変長配列	未対応
Enum	列挙体	未対応
String and Wide String	文字列、ワイド文字列	未対応
Fixed-Point Decimal Type	固定小数点数値	未対応
Value Types	値型	未対応
Pseudo-Object Types		
Type Code	型コード	未対応
Any	すべての型が格納可能な型	未対応
Exception	例外	未対応
Object Reference	オブジェクト参照	ObjectKey で代用
Abstract Interfaces	抽象インタフェース	未対応

2.4.3. ObjectKey

RTMSafety では、オブジェクトを一意に識別するための ID として ObjectKey を利用しています。

ObjectKey は、配置されたノードを示すための Node ID、プロトコル毎のチャンネルの区別を示す Channel ID、プロトコルの種類を示す Protocol ID、実行するタスクを示す Task ID、コンポーネントを一意に示す Component ID、ポートを一意に示す Port ID から構成されます。

特に、RT コンポーネントを特定するための ObjectKey を RTC ID、データポートを特定するための ObjectKey をデータポート ID、実行コンテキストを特定するための ObjectKey を実行コンテキスト ID と呼びます。

2.5. RTMSafety Bridge

RTMSafety は、RTMSafety 上で動作する RT コンポーネント（Safety コンポーネント）と、OpenRTM-aist-1.0 互換コンポーネントの橋渡しを行うためのツールです。

例えば、図 2-10 に示すような安全関連系で動作するアームロボットシステムと、非安全関連系で動作する音声認識システムを連携させて、音声認識によりロボットアームを操作するようなシステムを構築することができます。また、RTSystemEditor 上で、RTMSafety 上で動作する RT コンポーネント（以下、Safety コンポーネント）の状態や、RT コンポーネント間の接続状況を確認することができます。

ただし、このようなシステムを構築する場合は、非安全関連系で問題が発生しても、安全関連系に影響しないように対策を講じる必要があります。

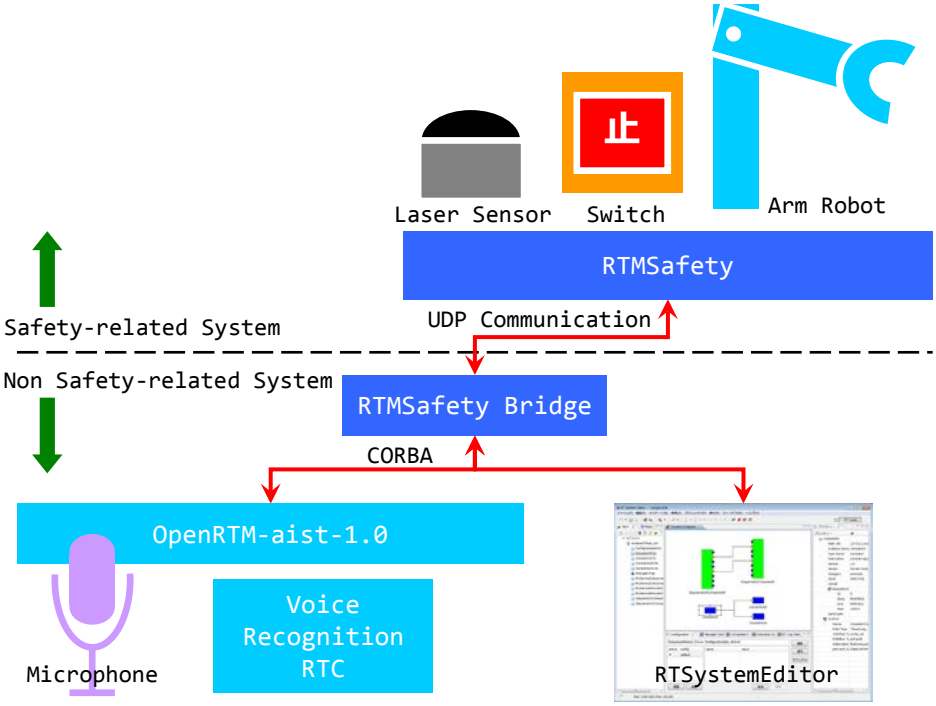

図 2-10　RTMSafety Bridge の利用イメージ

2.5.1. RTMSafety Bridge の構成

RTMSafety Bridge は、RTMSafety 上で動作する RT コンポーネント（Safety コンポーネント）と、OpenRTM-aist-1.0 互換コンポーネントの橋渡しを行うためのツールです。

RTMSafety Bridge の位置づけを図 2-11 に示します。

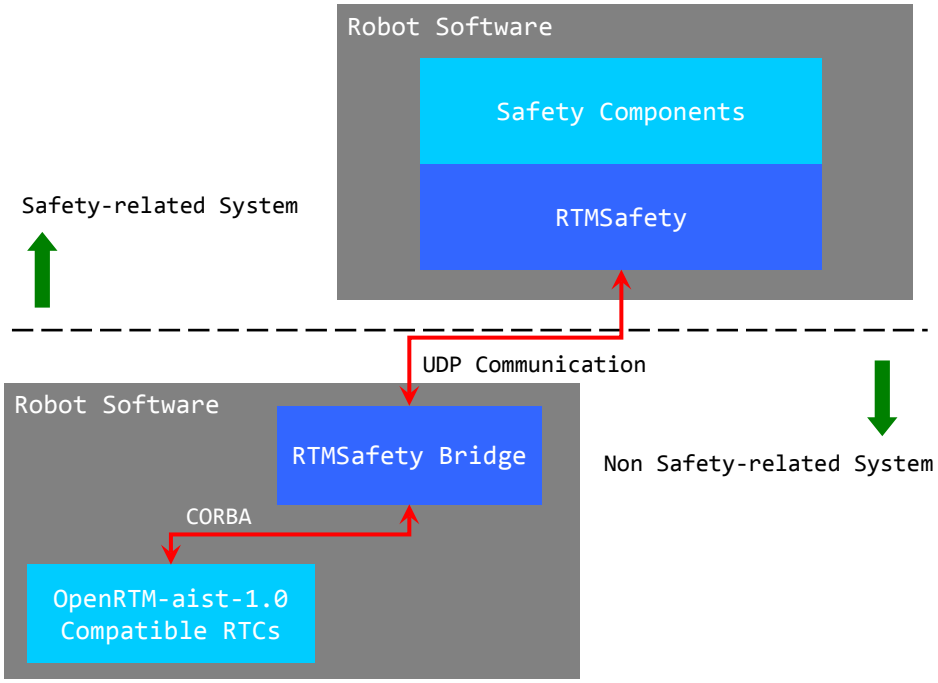

図 2-11　RTMSafety Bridge の位置づけ

RTMSafety と RTMSafety Bridge の間は UDP 通信、RTMSafety Bridge と OpenRTM-aist-1.0 互換コンポーネントの間は CORBA により通信を行います。RTMSafety Bridge では、この UDP と CORBA のプロトコル変換を行っています。

RTMSafety Bridge を利用することにより、安全関連系で動作する RT システムと、非安全関連系で動作する OpenRTM-aist-1.0 互換コンポーネントを連携させることが可能となります。これにより、既存資産である OpenRTM-aist-1.0 互換コンポーネントを活用することが可能となります。

2.5.2. RTMSafety Bridge の機能

RTMSafety Bridge が提供する機能は下記のとおりです。

- データポートを介した Safety コンポーネントとのデータの送受信
- Safety コンポーネントの状態の確認

RTMSafety Bridge の構成を図 2-12 に示します。

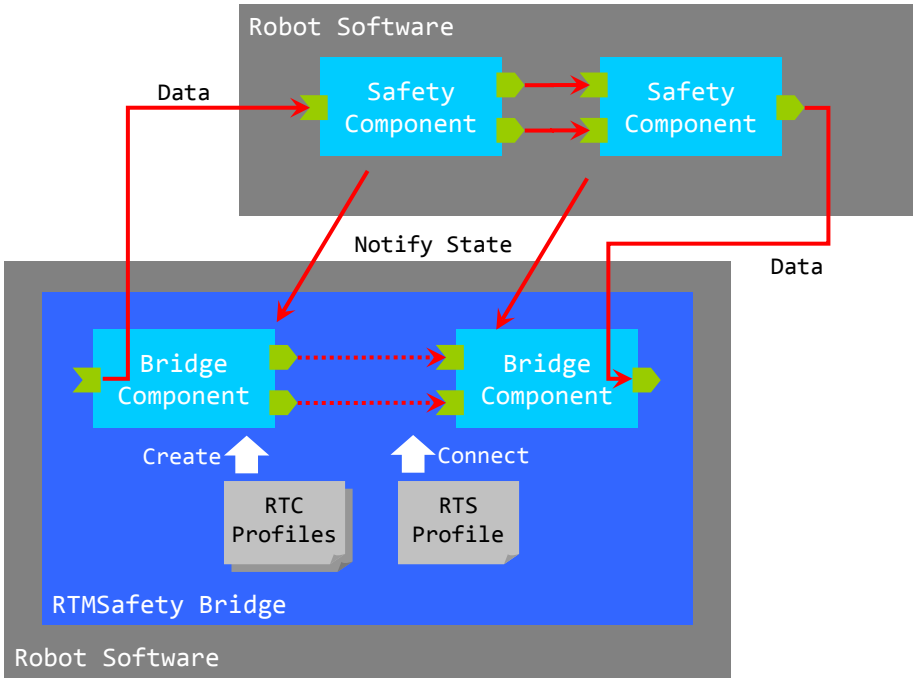

図 2-12　RTMSafety Bridge の構成

RTMSafety Bridge は、Safety コンポーネントを OpenRTM-aist-1.0 互換コンポーネントとして見せかけるための、Bridge コンポーネントを有しています。

Bridge コンポーネントは、Safety コンポーネントの射影であり、Bridge コンポーネントの InPort に書き込まれたデータは Safety コンポーネントの InPort に書き込まれ、Safety コンポーネントの OutPort から出力されたデータは Bridge コンポーネントの OutPort から出力されます。

また、Safety コンポーネントは常に Bridge コンポーネントに対して自身の状態を送信しており、Bridge コンポーネントの状態と常に同期されます。
さらに、Safety コンポーネント間の InPort と OutPort が接続されている場合、Bridge コンポーネント間の InPort と OutPort が接続されているように見せかけます。（実際には Bridge コンポーネント間のデータの送受信は行われません。）
ただし、Safety コンポーネントに対して activate_component や deactivate_component などのイベントを送信することはできません。RTMSafety ではシステムの安全性を考慮し、RT コンポーネントの外部からのイベント送信を禁止しているためです。

なお、Bridge コンポーネントの構成（データポートの数やコンポーネントの名前）は RTC Profile から生成されます。また、Bridge コンポーネントのデータポートの接続は、RTS Profile により自動的に接続されます。