OpenRTM-aist-1.0.0-RC1について

株式会社セック

2009年6月25日版

本資料の目的

- OpenRTM-aistの正式版のリリース候補であるOpenRTM-aist-1.0.0-RC1は、OpenRTM-aist-0.4.xから、いくつかの機能追加や仕様変更が行われています。
- 本資料は、OpenRTM-aist-1.0.0-RC1での変更点を明らかにし、OpenRTM-aist-1.0.0-RC1 を使ってRTコンポーネントを開発するためのノウハウや注意事項をまとめたものです。
- なお、本資料の内容は株式会社セックにて独自に調査・分析した結果であり、内容に誤りが含まれている可能性があります。本資料の内容に誤りを見つけた場合はRTMメーリングリスト（openrtm-users@m.aist.go.jp）までご連絡いただけると幸いです。

※なお、本資料の内容は、OpenRTM-aist-0.4.xの仕様を理解している方を対象として記述してあります。

Copyright© 2009 SEC Co., Ltd. All rights reserved.

概要

- 本資料では、OpenRTM-aist-1.0.0-RC1で追加・変更された以下の機能について説明を行います。
- OpenRTM-aist-1.0.0-RC1の変更点
  - IDLの変更
  - データポートのモデル変更
  - 複合コンポーネントの導入
  - コンフィギュレーションのコールバック
  - マネージャの機能追加
  - OS抽象化層(coil)
  - ログ機能
- ツールの変更点
  - RTCビルダ
  - RTシステムエディタ

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

IDLの変更

IDLの変更

- OMG RTC Specification 1.0に準拠。
- IDLが変更されているため、0.4のRTCと1.0のRTCを接続することはできない。
- 変更点
  - RTObject:get_owned_contexts/is_alive
  - EC:add_component/remove_component
  - Configuration:set_configuration_set
  - Port → PortService
  - PortStatus, InPortCdr, OutPortCdr

Copyright© 2009 SEC Co., Ltd. All rights reserved.

RTC::RTObject

- 実行コンテキストの取得
  - 0.4/1.0で利用可能
    - get_context(ec_id)
  - 1.0で廃止
    - get_contexts()
  - 1.0で追加
    - get_owned_contexts()
    - get_participanting_contexts()
    - get_context_handle(ec)

コンポーネントと同時に生成された実行コンテキストを取得

後で追加した実行コンテキストを取得

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

RTC::RTObject

- LifeCycleState
  - UNKNOWN_STATE → CREATED_STATE
- is_alive() → is_alive(ec)
  - 引数で渡した実行コンテキストが、owned_contextsかparticipanting_contextsに含まれる場合はtrueを返す

Copyright© 2009 SEC Co., Ltd. All rights reserved.

ExecutionContext/Configuration

- RTC::ExecutionContext
  - add/remove →
    add_component/remove_component
- SDOPackage::Configuration
  - set_configuration_set_values(id, config_set)
    →set_configuraation_set_values(config_set)

Copyright© 2009 SEC Co., Ltd. All rights reserved.

ポート

- RTC::Port → RTC::PortService
- データポート(OpenRTM独自仕様)
  - InPortAny/OutPortAny → InPortCdr/OutPortCdr
  - put/getの引数がanyからsequence<octet>に変更
  - put/getの戻り値がPortStatus

Copyright© 2009 SEC Co., Ltd. All rights reserved.

データポートのモデル変更

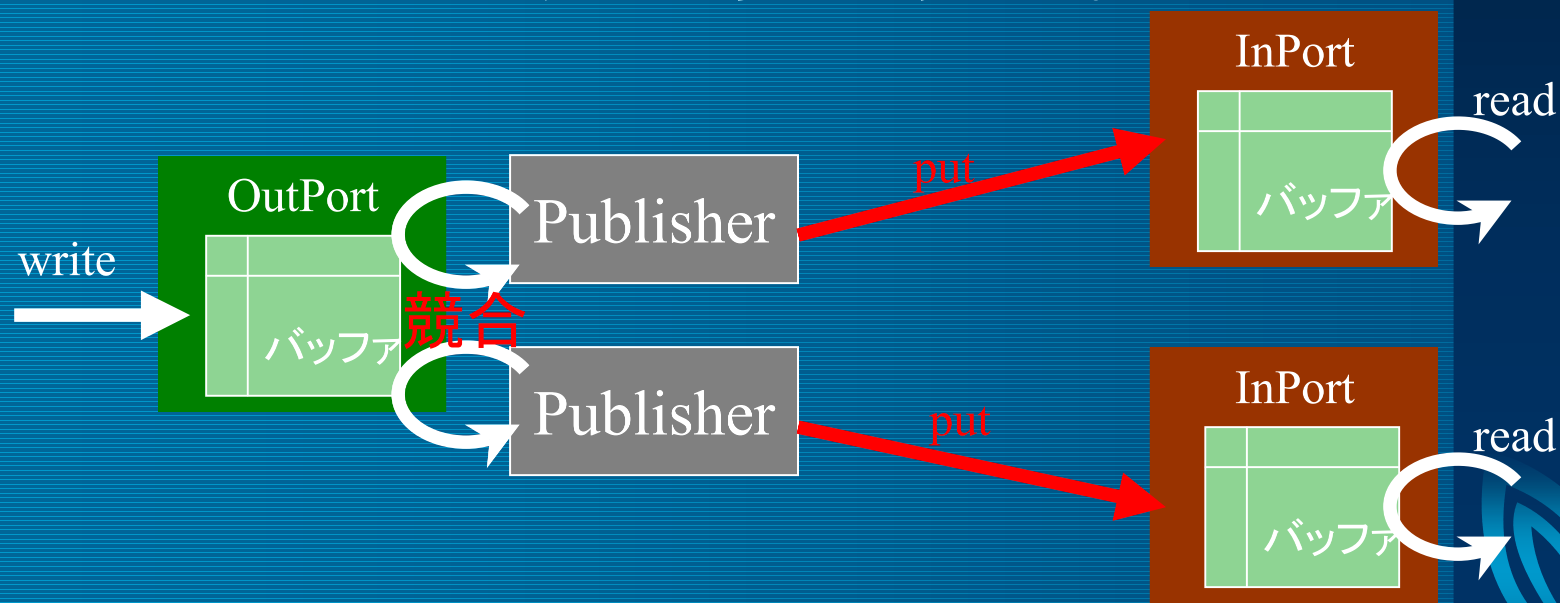
データポートのモデル変更

- 複数InPortへの分配ができない問題の解決
- バッファリングの指定
- InPortへの書き込みステータス
- データ形式としてCDRを利用
- PULL型接続による通信
- コールバックの追加
- ポートの活性化・非活性化
- その他

Copyrightⓒ 2009 SEC Co., Ltd. All rights reserved.

複数InPortへの分配(0.4モデル)

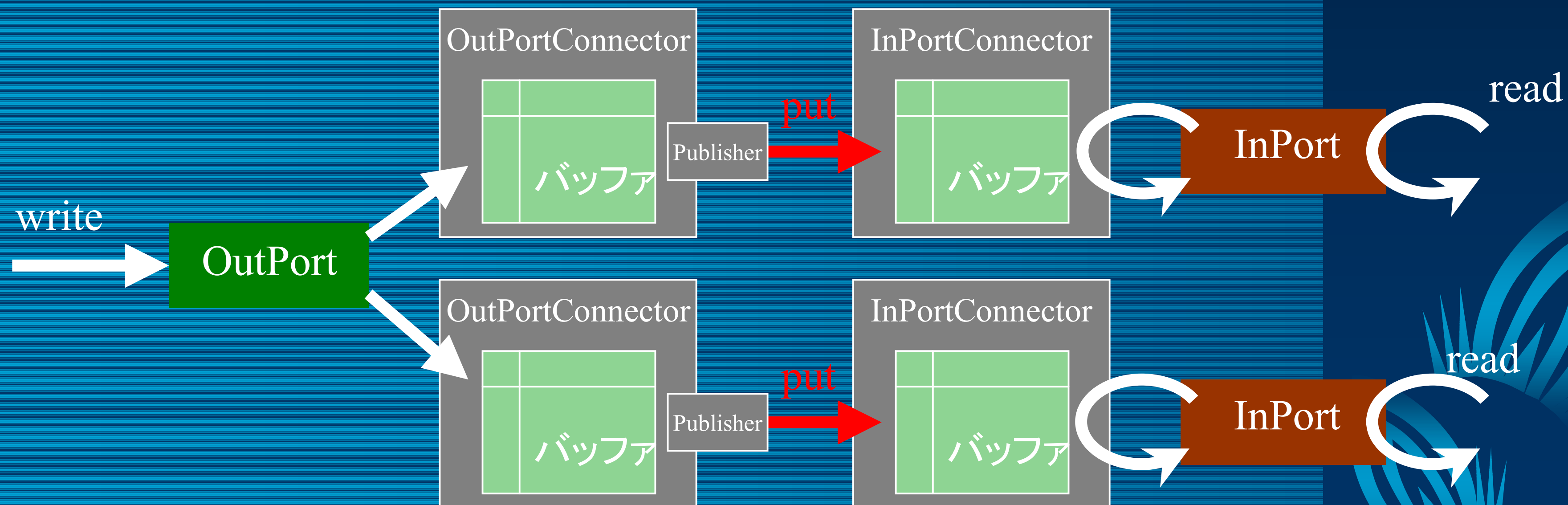
- OutPortのバッファが1つしかないため、いずれか1つのPublisherのみしかデータを取得できない。
- 複数に分配するには、OutPortのバッファをNullBufferにする必要があった。



Copyright© 2009 SEC Co., Ltd. All rights reserved.

複数InPortへの分配(1.0モデル)

- Connectorにバッファを持たせることにより、複数のInPortへの分配が可能に。



CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

バッファリングの指定

- 0.4では、InPortやOutPortの2番目の型パラメータにバッファ型を指定。
  - 例：InPort<TimedLong, RingBuffer>
  - OutPort<TimedLong, NullBuffer>
- 1.0では、接続時のパラメータとして、バッファ型を指定。型パラメータでの指定は不可。
  - buffer_type
  - デフォルトではRingBuffer

Copyright© 2009 SEC Co., Ltd. All rights reserved.

InPortへの書き込みステータス

- 0.4ではInPortへの書き込みの成否が不明。
- 1.0では、書き込みステータスが通知される。
  - PORT_OK
  - PORT_ERROR
  - BUFFER_FULL
  - BUFFER_EMPTY
  - BUFFER_TIMEOUT
  - UNKNOWN_ERROR
- ただし、ユーザーのRTコンポーネントからは、boolでしかステータスをとれない。
- 通信に問題が発生した場合、自動的に切断処理が行われる。

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

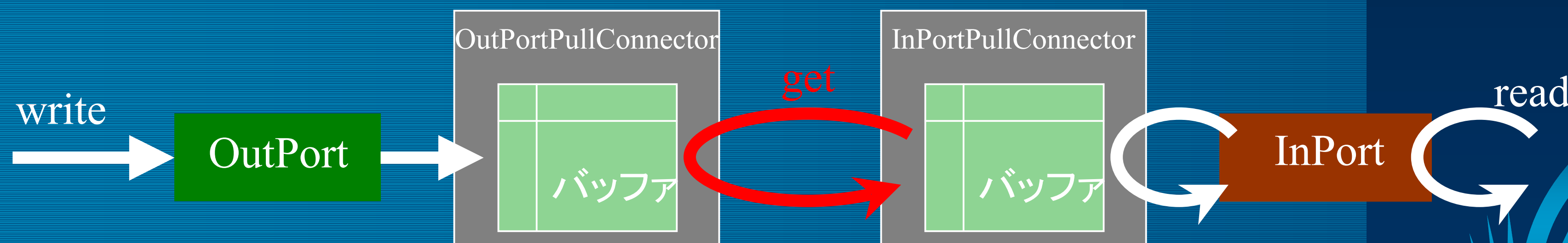
データ形式としてCDRを利用

- 0.4では、データポート間のデータ形式にAny型が利用されていた。
- 1.0では、CDR形式のバイナリ配列を送信する。
- CDR(Common Data Representation)とは
  - CORBAでデータを送信する際のフォーマット。
  - Any型も内部的にはCDRに変換されて送信されている。
- 何が変わるか
  - RTC開発者、利用者には何も変わらない。
  - RTCデバッガやRTCハンドルなどのツールで対応が必要。
  - Any型への変換処理が減るため、オーバーヘッドが小さくなる。
  - 型情報が付与されないため、データサイズが小さくなる。
  - 共有メモリや他の通信方式のデータポートが作りやすくなる。

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

PULL型接続による通信

- InPortをreadすると、能動的にOutPortからデータを取得する接続方法。
- 1.0.0-RC1では動作しない？
- 多重接続が行われた場合の挙動は？



Copyrightⓒ 2009 SEC Co., Ltd. All rights reserved.

コールバックの追加

- 接続と切断の通知が行われる。
  - onConnect
  - onDisconnect
- InPortのコールバックはonRead、onReadConvertのみ実装。onWriteなどの実装はペンディング。
- OutPortのコールバックはすべて実装済み。

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

ポートの活性化・非活性化

- activateInterfaces/deactivateInterfacesというインタフェースがポートに追加された。
- コンポーネントが活性化・非活性化されたタイミングでポートも活性化・非活性化される。
- データポートとサービスポート両方とも。
- サービスポートは、接続後に非活性化→活性化すると通信ができない問題がある。

Copyright© 2009 SEC Co., Ltd. All rights reserved.

その他

- IORによるリファレンス交換
  - 0.4では、ポートの情報をオブジェクトリファレンスでやりとりしていた。
  - 1.0では、ポートの情報をIOR(文字列)でやりとりすることが可能。
- OutPortのwrite時にタイムスタンプを付与。
  - 1.0では、OutPortのwrite時にデータにタイムスタンプが自動で付与されるようになった。
  - データポートで扱うデータ形式は、必ずTime型のtmというメンバを持つ必要がある。

Copyright© 2009 SEC Co., Ltd. All rights reserved.

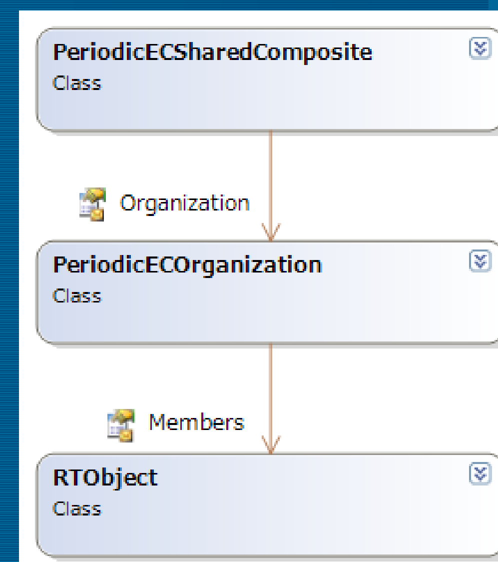
複合コンポーネントの導入

複合コンポーネント

- RTCを入れ子構造にする仕組み。
- コンポーネントの実装ではなく、実行時に実現する。
- RTSystemEditorのドラッグアンドドロップ操作で複合化することができる。

Copyright© 2009 SEC Co., Ltd. All rights reserved.

クラス構成



- PeriodicECSharedComposite
  - RTObject_implを継承。
  - 親RTC(入れ物)役のコンポーネント。
  - 公開するポート情報をConfigurationとして持つ。
- PeriodicECOrganization
  - SDO::Organizationを継承。
  - メンバーとして、子RTCを複数持つ。

Copyright© 2009 SEC Co., Ltd. All rights reserved.

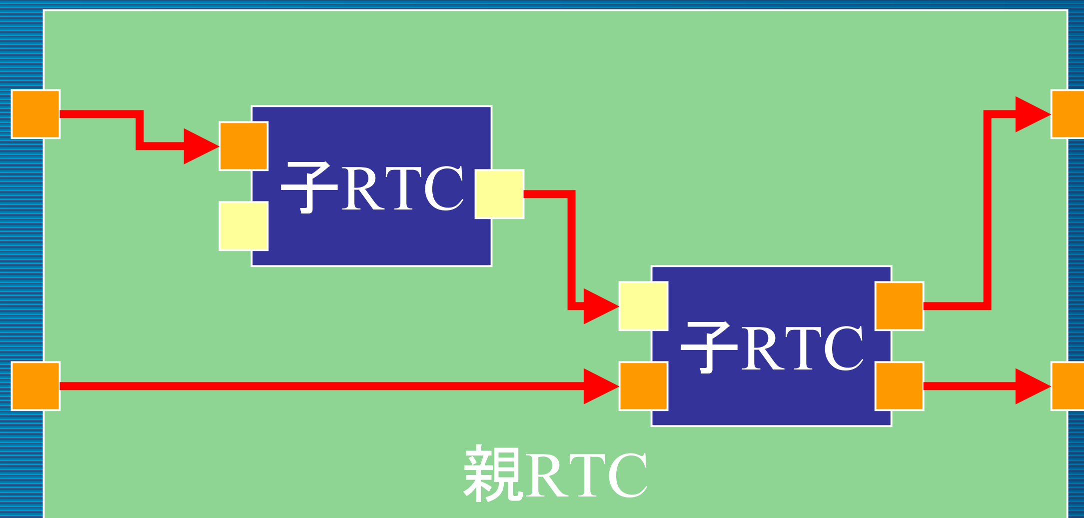
複合化シーケンス

- Manager経由で、PeriodicECSharedCompositeコンポーネントを生成する。
- 生成したPeriodicECSharedCompositeからget_organizationでPeriodicECOrganizationを取得する。
- Organizationにadd_membersで子となるRTCを設定する。
- 子のRTCの実行コンテキストをstopし、PeriodicECSharedCompositeの実行コンテキストを関連付ける。

Copyright© 2009 SEC Co., Ltd. All rights reserved.

ポートの委譲と隠蔽

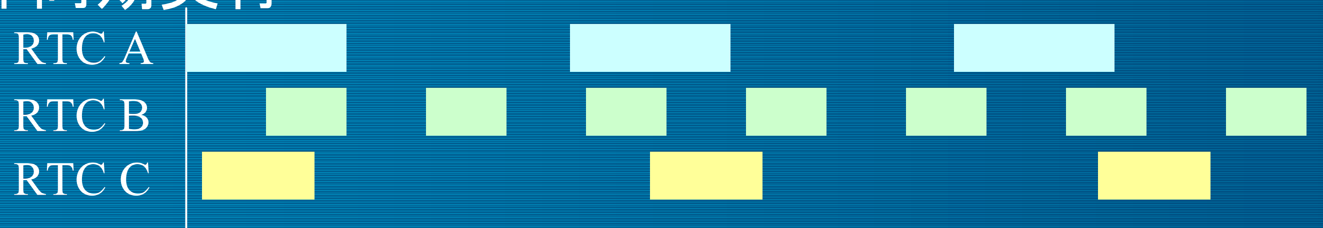
- 子RTCのポートを親RTCに委譲することができる。
- PeriodicECSharedCompositeのコンフィギュレーション(conf.default.exported_ports)に設定。



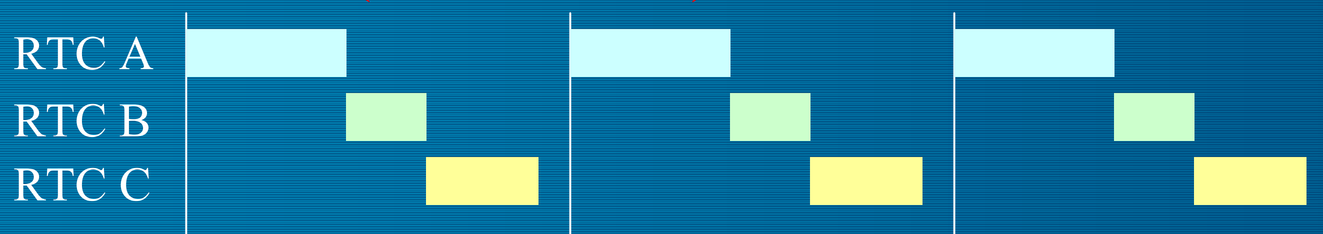
Copyright© 2009 SEC Co., Ltd. All rights reserved.

実行の同期･非同期、並列･直列

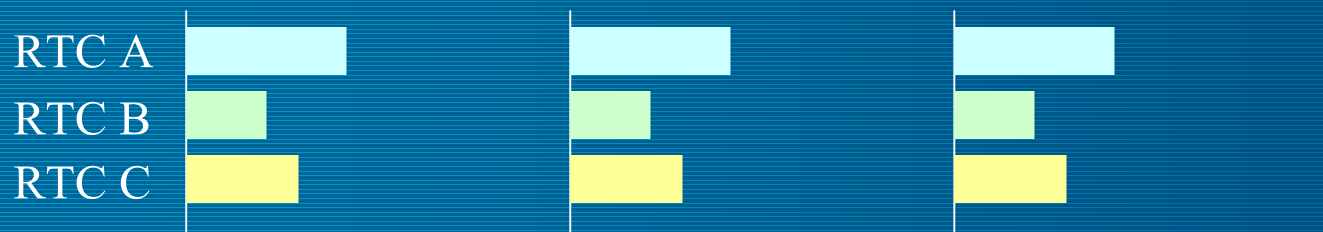
非同期実行



同期直列実行(現状の実装)



同期並列実行



CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

コンフィギュレーションのコールバック

コンフィギュレーションのコールバック

- パラメータの更新や、追加・削除などをしたときにコールバックが呼び出される。
  - onUpdate
  - onUpdateParam
  - onSetConfigurationSet
  - onAddConfigurationSet
  - onRemoveConfigurationSet
  - onActivateConfigurationSet

Copyright© 2009 SEC Co., Ltd. All rights reserved.

マネージャの機能追加

マネージャ

- 新機能
  - マネージャがCORBAサーバントに。
  - オブジェクトリファレンスをネームサーバに登録。
  - リモートでコンポーネントを生成できる。
- 未実装の機能
  - 新しいプロセスとしてマネージャを生成する。
  - ホストごとにマネージャを階層化する。
  - コンポーネントをリモートでロードする。

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

OS抽象化層(coil)

coilとは

- Common Operatingsystem Infrastructure Layerの略
- OpenRTM-aistに必要な機能を実装
- 移植性向上
  - OpenRTM-aist-0.4まではACEが要インストール
  - オプションの指定によりACEが利用可能
- OSの抽象化
  - 以下のプラットフォームに対応
    - 公開　　　POSIX，Win32，ACE
    - 非公開　　TOPPERS，VxWorks
- ライブラリサイズが小さい。(ACE=1MB, coil=162kB)

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil機能一覧(クラス定義あり)

- Allocator
- Condition
- DynamicLib
- Mutex
- PeriodicTask
- Properties
- Signal
- Task
- Timer
- TimeMeasure
- TimeValue
- UUID
- Async
- Factory
- Guard
- Listener
- Logger
- NonCopyable
- OS
- Singleton
- memory

Copyright© 2009 SEC Co., Ltd. All rights reserved.

coil機能一覧(クラス定義なし)

- stringutil
- atomic
- Time
- File

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Singleton

- シングルトンパターンを実現するテンプレートクラス
- 継承して利用

使用方法:

```
class SampleSingleton :
        public coil::Singleton<SampleSingleton> {
};
```

SampleSingleton::instance(); としてアクセス

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Allocator

- メモリの割り当て・解放を自前で用意するための仕組みを提供
- 現状、実装途中。
- シングルトンパターンで実装されている

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Mutex

- ロック機構を提供するクラス
- 実行文単位のクリティカルセクションのアトミック性を保証

スレッド1：

```
func1(){
    mutex->lock();
    クリティカルセクション1
    mutex->unlock();
}
```

スレッド2：

```
func2(){
    mutex->lock();
    クリティカルセクション2
    mutex->unlock();
}
```


CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Guard

- Mutexクラスを使いやすくするためのクラス
- スコープ単位のクリティカルセクションのアトミック性を保証
- ロックの解放忘れを防ぐことができる

スレッド1：
```
func1(){
    guard(mutex);
    クリティカルセクション1
}
```

スレッド2：
```
func2(){
    guard(mutex);
    クリティカルセクション2
}
```

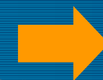

Copyright© 2009 SEC Co., Ltd. All rights reserved.

coil::Condition

- イベントの発生を待機中のスレッドに通知する仕組み
- wait()はタイムアウト時間を指定可能

```
スレッド1：
func1(){
    スレッド2生成
    cond->wait();
    処理1-1
}
```

```
スレッド2：
func2(){
    処理2-1
    cond->signal();
    処理2-2
}
```



処理2-1の後に
処理1-1が実行
される

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Factory

- 機能拡張のための仕組みを提供するクラス
- ファクトリパターンを拡張
- ファクトリの登録を行い、オブジェクトを作成
- ファクトリの抹消・オブジェクトの削除も可能

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Task

- クラス単位でスレッドを生成するための機能
- Taskクラスを継承し、svcメソッドにスレッドの処理を実装する

```
class SampleTask : public coil::Task {
  public:
    void svc() {…}  // スレッドで実行されるメソッドをオーバライド
    void create() {
      activate();  // スレッドを作成するメソッドを呼び出す。
    }
};
```


CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Async

- メソッドを非同期実行するための機能
- invoke()で非同期呼び出し、finished()で実行完了を確認
- coil::Taskクラスを継承

Copyrightⓒ 2009 SEC Co., Ltd. All rights reserved.

coil::PeriodicTask

- 新規スレッドにより、処理を周期的に実行
- PeriodicTaskBaseクラスを継承
  – PeriodicTaskBaseクラスはcoil::Taskクラスを継承

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Listener

- イベント発生時に呼出されるコールバック関数を設定する

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Timer

- 周期的にコールバック関数を実行
  - coil::Listenerテンプレートクラスを利用

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::TimeMeasure

- 複数回計測した時間の統計を計算
  - 最小
  - 最大
  - 平均
  - 標準偏差
- tick()からtack()までの時間を計測

Copyrightⓒ 2009 SEC Co., Ltd. All rights reserved.

coil::TimeValue

- マイクロ秒単位で時間データを管理
- double型との相互変換が可能

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::UUID

- 重複がないID(UUID:Universal Unique IDentifier)の生成が可能
- 128bitの数値 or 36文字の文字列で表現
- MACアドレス、時間、乱数などを元に、ユニークなIDを生成する
- 実装によっては、短時間に連続生成すると、同じIDが振られることがある？

Copyright© 2009 SEC Co., Ltd. All rights reserved.

coil::Properties

- キーと値のセットを階層化して管理するための機能
- FileStreamなどstreamとの相互変換が可能
- JavaのPropertiesと同等の機能を提供
- コンポジットパターンを利用
  - 親が1つ、子が複数

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Signal

- Ctrl+Cなどの指定シグナルをキャッチし、対応するアクションを実行

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::Logger

- ログ出力に関する機能を提供
- OpenRTM-aist-1.0.0-RC1のログ出力機能RTC::Loggerは、本機能を利用

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::NonCopyable

- オブジェクトのコピーを禁止するために利用
- private継承して利用
- コピーコンストラクタと=operatorを利用すると、コンパイルエラーになる

使用方法:

```
class SampleNonCopyable :
        private coil::NonCopyable {
    …
};
```

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::memory

- スマートポインタ機能。現状、実装途中。
- 参照回数をカウントし、不要になったメモリを自動で解放する
- std::tr1::shared_ptrを使用するか、coil::shared_ptrを使用するかの選択が可能
  - std::tr1::shared_ptrが使用可能なのは、コンパイラに TR1がある場合

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::OS

- coil:: GetOptクラス
  - コマンドライン引数の取得
- uname()
  - OS情報の取得
- getpid()
  - プロセスIDの取得
- getenv()
  - 環境変数の取得

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

coil::DynamicLib

- 共有ライブラリを動的にロードするための機能
- Linuxの場合は*.soファイル、Windowsの場合は*.dllファイルを扱う

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

クラス定義なし

- stringutil
  - 文字列操作
- atomic
  - アトミック性を保障した加算・減算処理
- Time
  - 現在時刻の取得・設定機能、秒単位・マイクロ秒単位のスリープ機能
- File
  - ディレクトリ名・ベース名の抽出

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

ログ機能

ログ機能

- プロパティの設定
  - プロパティの設定方法
  - プロパティのkey名とvalue候補
- 利用方法
  - ソースコード上の作法
  - ログ出力用マクロ一覧
- ログのフォーマット
- 注意点

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

プロパティの設定方法

- RTコンポーネントが起動時に読み込むコンフィギュレーションファイル(ex. rtc.conf)に、ログ制御用のプロパティを設定する。

rtc.conf設定例：

```
exec_cxt.periodic.rate:1.0
corba.nameservers: localhost
logger.enable: YES
logger.log_level: TRACE
logger.file_name: rtc.log
manager.modules.load_path: ./
manager.modules.preload: Sample.so
Hoge.Sample.config_file: Sample.conf
```


Copyright© 2009 SEC Co., Ltd. All rights reserved.

プロパティのkey名とvalue候補

- ログ出力のOn/Off
  - logger.enable: [YES / NO]
- ログ出力レベルの指定
  - logger.log_level: [SILENT / FATAL / ERROR / WARN / INFO / DEBUG / TRACE / VERBOSE / PARANOID]（SILENTを除く、指定したレベルから左側のログを出力する）
- ログ出力先の指定
  - logger.file_name: [STDOUT / 出力先ファイル名]（STDOUT=標準出力）

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

ソースコード上の作法

- ログ機能を必要とするソースファイル(*.h, *.cpp)に以下のインクルード指定を追加する。
  - #include <rtm/SystemLogger.h>
- マネージャの初期化後に利用可能。
- ログ出力を必要とする位置にマクロを組み込む。
  - 例:

```
RTC::ReturnCode_t Sample::onExecute(RTC::UniqueId ec_id)
{
    RTC_TRACE( ( "%s", __FUNCTION__ ) );
    std::string target;
    RTC_TRACE_STR( target );
    return RTC::RTC_OK;
}
```


Copyright© 2009 SEC Co., Ltd. All rights reserved.

ログ出力用マクロ一覧

- RTC_LOG(出力レベル, 出力フォーマット文字列)
- RTC_LOG_STR(出力レベル, 文字列)
- RTC_FATAL(出力フォーマット文字列)
- RTC_FATAL_STR(文字列)
- RTC_ERROR(出力フォーマット文字列)
- RTC_ERROR_STR(文字列)
- RTC_WARN(出力フォーマット文字列)
- RTC_WARN_STR(文字列)
- RTC_INFO(出力フォーマット文字列)
- RTC_INFO_STR(文字列)
- RTC_DEBUG(出力フォーマット文字列)
- RTC_DEBUG_STR(文字列)
- RTC_TRACE(出力フォーマット文字列)
- RTC_TRACE_STR(文字列)
- RTC_VERBOSE(出力フォーマット文字列)
- RTC_VERBOSE_STR(文字列)
- RTC_PARANOID(出力フォーマット文字列)
- RTC_PARANOID_STR(文字列)

ログレベルに対応した各マクロは
対応する出力レベルを指定して
RTC_LOG / RTC_LOG_STRを
呼び出している。

Copyright© 2009 SEC Co., Ltd. All rights reserved.

ログのフォーマット

- ログ出力の基本フォーマットを以下にまとめる。
  - 出力例:
    Jun 15 16:27:58 TRACE: manager: Manager::getORB()

月 日 時:分:秒 ログレベル: 出力元: マクロに指定した文字列

  - RTCのコンストラクタでは、出力元が出力されない。
  - RTCのデストラクタは、ログが出力されない。
  - 出力元はコンポーネントのインスタンス名。

Copyright© 2009 SEC Co., Ltd. All rights reserved.

ログ機能利用の注意点

- 出力フォーマット文字列を指定する(_STRのない)マクロを利用する場合、変換指定子を含む文字列と、変換対象を()でまとめる。
  - × RTC_TRACE("%s", __FUNCTION__); (=ビルド出来ない)
  - ○ RTC_TRACE(("%s", __FUNCTION__));
  - 理由：
    出力フォーマット文字列を変換する関数[coil::sprintf]は可変引数を持つ。ログ出力マクロを固定引数として定義するため(?)に、可変引数部分を()でまとめ、１つの引数としている。マクロ内では、coil::sprintfを()無しで利用し、プリプロセッサで展開後、マクロの引数の()を受けて、関数呼び出しとしてのコードが成立するようになっている。

```
#define RTC_LOG(LV, fmt) ¥
    ::coil::sprintf fmt
```


Copyright© 2009 SEC Co., Ltd. All rights reserved.

ログ機能利用の注意点

- _STRのあるマクロを利用する場合、std::basic_ostreamへの左シフト演算子(<<)を持つデータ型を指定する。

```
std::string target;
RTC_TRACE_STR( target );
```

  - 理由：
    マクロ内で、マクロ引数をstd::basic_ostreamへ左シフト演算子で挿入しているため。

- OpenRTM-aist-1.0.0-RC1のログも出力される。（システムへのRTCの関わり方を確認するには向いているが、RTC内部のロジック確認には不向き。）

Copyright© 2009 SEC Co., Ltd. All rights reserved.

ログ機能:その他

- NO_LOGGINGを定義することで、ログを出力させないことが可能。
- 要望
  - システムログとユーザーログを分離させたい。
  - ログファイルが一定のサイズを超えたら、別のファイルに書き込むようにしたい。
  - ログのフィルタリング機能が欲しい。

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

RTCビルダ

0.4.2版RTCビルダからの変更点

- インタフェース全般の見直し
  - ヒント項目の追加
  - アクティビティ・セクションの独立
- OpenRTM-aist 1.0.0 RC1に対応（C++のみ）
- RtcProfileがver.0.1からver.0.2へ変更
- プログラミング言語の追加（C#、VB.NET）
- コンポーネント用設定ファイルの生成

Copyright© 2009 SEC Co., Ltd. All rights reserved.

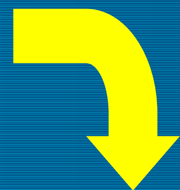
インターフェース全般の見直し
ーヒント項目の追加

ver.0.4.2

Module description :	ModuleDescription
Module version :	1.0.0
Module vender :	VenderName
Module category :	Category
Component type :	STATIC
Component's activity type :	PERIODIC
Component kind :	DataFlowComponent
Number of maximum instance :	1
Execution type :	PeriodicExecutionContext
Execution Rate :	1.0

Output Project

test1



ver.1.0.0 RC1

ヒント項目の追加

Copyright© 2009 SEC Co., Ltd. All rights reserved.

インターフェース全般の見直し
ーアクティビティ･セクションの独立

ver.0.4.2

動作概要:

事前条件:

事後条件:

on_finalize

Implemented

基本 | データポート | サービスポート | コンフィギュレーション | 言語・環境 | RTC.xml | ドキュメント生成

0.4.2版では、
ドキュメント生成･
セクションで選択

1.0.0 RC1版では、
アクティビティ･
セクションに独立

ver.1.0.0 RC1

このセクションでは使用するアクションコールバックを指定します。

コンポーネントの初期化と終了処理に関するアクション

on_initialize
on_finalize

実行コンテキストの起動と停止に関するアクション

on_startup
on_shutdown

alive状態でのコンポーネントアクション

on_activated
on_deactivated
on_aborting
on_error
on_reset

Dataflow型コンポーネントのアクション

on_execute
on_state_update
on_rate_changed

FSM型コンポーネントのアクション

on_action

Mode型コンポーネントのアクション

基本 | アクティビティ | データポート | サービスポート | コンフィギュレーション | ドキュメント生成 | 言語・環境 | RTC.xml

Copyright© 2009 SEC Co., Ltd. All rights reserved.

OpenRTM-aist 1.0.0 RC1に対応

- OpenRTM-aist 1.0.0 RC1の雛形を生成（C++のみ。それ以外は0.4の雛形）
- OpenRTM-aist 0.4向けの雛形とはソースコードレベルでの相違点は見られない
- ビルド時にリンクするライブラリなどは1.0.0 RC1のものを利用

Copyrightⓒ 2009 SEC Co., Ltd. All rights reserved.

RtcProfileがver.0.1からver.0.2へ変更

- ソースコード生成時にRtcProfile（RTC.xml）というファイルが生成される。
- ver.0.1とver.0.2とでは互換性なし（1.0.0 RC1版のRTC Builderでは、0.4.2版で生成したRtcProfileが読込めない）

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

プログラミング言語の追加

- C#とVB.NETに新規対応
- コンポーネントの開発・動作にはOpenRTM.NET 0.4.0のインストールが必要

- OpenRTM.NET 0.4.0のダウンロードサイト
  – http://book.mycom.co.jp/support/e1/rt_middleware/
- OpenRTM.NET 1.0.1のダウンロードサイト
  – http://tedia.jp/member/download/detail.php?id=161

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

コンポーネント用設定ファイルの生成

- コンフィギュレーション・パラメータが設定されている場合に、コンポーネント用設定ファイルを自動生成
- 設定ファイル名：[コンポーネント名].conf
- RTSystemEditorでのコンフィギュレーションの編集で、ウィジェットや制約条件に反映

Copyright© 2009 SEC Co., Ltd. All rights reserved.

RTCビルダ1.0の注意点

- コード生成での出力選択時、独自に実装したコードがマージの対象外（消滅する）
- C++ヘッダファイル中、onInitializeのコメントがコンフィギュレーションの有無によって変化
  - コンフィギュレーションなし：/*!　*/
  - コンフィギュレーションあり：/**　*/
- 未実装機能
  - on_actionおよびon_mode_changeアクティビティ

Copyrightⓒ 2009 SEC Co., Ltd. All rights reserved.

RTシステムエディタ

0.4.2版RTシステムエディタとの変更点

- 複合コンポーネントの機能
- コンフィギュレーション編集の操作性向上
- マネージャビューの追加

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

複合コンポーネントの機能

- コンテキストメニューから「Create Composite Component」を選択することで、複合コンポーネントを生成できる。
- 複合コンポーネントをダブルクリックすることで、コンポーネントの内部(子RTC)を表示することができる。
- 内部を表示した状態で、ネーミングサービスからRTCをドラッグすると、子RTCを追加することができる。

Copyrightⓒ 2009 SEC Co., Ltd. All rights reserved.

コンフィギュレーション編集の操作性向上

- コンフィギュレーションセットを複製する機能の追加
- コンフィギュレーションの編集用ダイアログを追加し、パラメータの編集方法として以下をサポート
  - テキストボックス(既存と同じ)
  - スライダ
  - スピナ
  - ラジオボタン
- コンフィギュレーションの制約条件チェックの機能を追加

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

コンフィグファイルの記述

- *コンポーネント名*.confファイルに、以下の記述を追加することで、ウィジェットのタイプや制約条件を指定できる。
- RTシステムエディタ上のコンフィギュレーションパラメータとしては表示されない。

conf._widget_.*変数名*: *ウィジェットタイプ*
conf._default.*変数名*: *制約条件*

ウィジェットタイプ記述例:

text	テキストボックス
spin	矢印付テキストボックス
slider.100	スライダ(分割数指定)
radio	ラジオボタン

制約条件記述例:

100	固定値
0<=x<100	範囲指定
(10,20,30)	列挙指定

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

マネージャビューの追加

- マネージャコンポーネントの持つ機能をGUI操作で管理可能
- 現在はCreate Componentのみが利用可能。
- 将来的には、モジュールのロード、マネージャプロセスの追加・削除などの機能が利用できるようになる。

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.

RTシステムエディタ1.0の注意点

- 複合コンポーネントを含むシステム状態を復元すると、復元に失敗
  - 成功させるためには、あらかじめ手作業で複合コンポーネントを作成しなければならない
- マネージャコンポーネントは、コンポーネントの起動毎に新しく作成されるものを利用？
  - コンポーネントを起動すると、古いマネージャコンポーネントで管理する情報が確認できなくなる
  - 最後に起動したコンポーネントを終了すると、他のコンポーネントの起動に関わらず、マネージャコンポーネントが終了したように見える

CopyrightⒸ 2009 SEC Co., Ltd. All rights reserved.