RTコンポーネントはどのように作ればよいか？

安藤慶昭（産総研），○栗原眞二（産総研），
ビグズジェフ（産総研），神徳徹雄（産総研）

目次

- コンポーネント開発
  - コールバックを実装する際のルール
  - データポートを利用する際に注意すべき点
  - サービスポートとそのインターフェースを実装する際の注意
  - コンフィギュレーション機能を利用する際に注意すべき点
- システム開発
  - システム全体をモジュール分割する際の粒度は？
  - モジュール間の依存性をどのように整理する？
- RTC作成に関する心得
- まとめ

コンポーネント開発（1）

コールバックを実装する際のルール

- 初期化・終了に関するルール
  - 動的リソースの確保をコールバックで行う場合は，確保をonInitialize，解放をonFinalize をできるだけ対にして行う．
  - デバイスのオープン，その他リソースの確保は可能な限りonInitialize で処理する．
- 実行に関するルール
  - 通常のコンポーネント(Dataow 型) では，主たるロジックはonExecute に実装する．データポートのコールバック等に主ロジックを実装することは推奨されない．
  - コンポーネントアクティビティ型(Component's activity type) は通常複合化が容易なPERIODIC 型が推奨される．その際onExecute の実行は十分短い時間かつ一定時間以内に終了するロジックとして実装する．

    onExecute 内では入力待ちなど，完全に停止するようなロジックを実装することは可能な限り避ける！
- エラーに関するルール
  - onExecute 中で回復可能なエラーは，エラー状態にはせずできるだけエラーが出た時点で回復するようなロジックにする．
  - onExecute に対してはonError，onActivated/onDeactivated に対してはonAbortingを対応させて実装する．

コンポーネント開発（2）

データポートを利用する際に注意すべき点

- データポートにはできるだけ既存の型を利用する．

  OpenRTM-aist-1.0 以降は，ExtendedDataTypes.idl, InterfaceDataTypes.idl 内で定義される型を利用することが推奨される
- InPort からは常にデータが取得できるとは限らない前提でロジックを実装する．（InPortのisNew()を使うなど。）
- onWriteやonReadなど,データポートのコールバックは，補助的に利用し主たるロジックを実行しない．
- 一つのポートからは一つの意味をもったデータのみ入出力するようにする．コンフィギュレーションなどでデータポートの意味を変えることは推奨されない．

コンポーネント開発(3)

サービスポートとそのインターフェースを実装する際の注意点

- サービスポートのインターフェースにはできる限り既存のものを利用する.
  OpenRTM-aist-1.0 以降は，ExtendedDataTypes.idl, InterfaceDataTypes.idl 内で定義される型を利用することが推奨される
- コンシューマを利用する場合は常に，プロバイダが対応付けられているとは限らない前提でロジックを記述し，利用する個所には必ず例外処理(C++ではtry-catch 節) を設ける.
- コンシューマからプロバイダを利用する場合，オペレーション呼び出しの実行時間が長くなる場合または不確定な場合，可能な限り非同期呼び出し(coil::Asyncを利用する等) を行う.
- サービスポートのコンシューマ，プロバイダはCORBA のオペレーション呼び出し規則に従い，C++等ではvar 型を用いるなどしてメモリリークが起こらないよう注意し実装する.

コンポーネント開発(4)

コンフィギュレーション機能を利用する際に注意すべき点

- コンポーネント内の変更される可能性のあるパラメータは，できるだけコンフィギュレーションパラメータにする.
- 型がある言語の場合，パラメータには適切な型を選択する. 連続値にはdouble，数・個数・順序を表現する場合はint，配列・行列には配列コンテナ型(vector)等を利用する.
- enum に相当する列挙型はstring 型を利用し，string からenum (int) への変換関数を定義したうえで，(C++の場合) bindParameter 関数の第4引数にこれを与える.

システム開発(1)

システム全体をモジュール分割する際の粒度は？

- 既存のシステムがある場合，最初は無理に分割せずに2 つのコンポーネントから始める．その後は拡張が必要な部分から徐々にモジュール化していく．
- 細粒度のモジュールは一般に再利用性が高い．しかし，分割したモジュール間に強固な依存性(例えばエンコーダ(とカウンタ) とモータ(とドライバ) は物理的に結合している等) がある場合には分割しても再利用できる可能性は低くくなるので注意すべきである．
- システムをモジュール分割していくと，異なるレイヤにまたがる比較的大きなモジュールがどうしても必要になる場合がある．これを無理に分割しようとしない．その部分はシステムの本質(デザインルール) である可能性がある．

モジュール間の依存関係ができるだけ少なくなるようにモジュールを分割した方がよい

システム開発(2)

モジュール間の依存性をどのように整理する？

- 密結合が必要な場合，データの流れとロジックの実行順序を検討のうえで，複合コンポーネント，実行コンテキストの共有による同期実行が必要ないか検討する．
- リアルタイム実行が必要な部分では，予め各コンポーネントの実行時間と周期を勘案したうえで，適切なリアルタイム実行コンテキストを利用する．
- より複雑な実行方法が必要な場合，実行コンテキストの拡張を検討する．

モジュール間の結合度が強い場合は、適切な実行コンテキストの選定が必要！

RTC作成に関する心得

- コンポーネントを作る前に似たような機能のコンポーネントがないかどうか調べ，可能な限り再利用する．問題があれば作者にフィードバックをする．
- ーから実装せずに既存のライブラリをできるだけ活用する．すでに公開され，使用されているコードは，何度も実行されているはずで，それだけでこれから書こうとするコードより信頼性は高い．
- OpenRTM-aist のクラスリファレンスマニュアルを活用する．サンプルコンポーネント以上のことを行うには，クラスリファレンスを参考にする．
  RTObject,InPort, OutPort クラスとそのスーパークラスのマニュアルを読めば何ができて何ができないか分かる．

サンプルコンポーネント以上のことを行うには，クラスリファレンスを参考にする！

まとめ

- RT コンポーネントでロボットシステムを作る，かつ作成されたモジュールをユーザ間で共有し，再利用するといった目的のためには，ユーザ間で共通したルールに従ってコンポーネントを作成する必要がある．
- ここに示したルールは，適用アプリケーション，ミドルウエアの機能や，その他の外的要因により変わることがあり，盲目的に従うのではなく常に検討を繰り返していく必要がある．
- ユーザコミュニティでこうした議論についても継続的に行われることを期待したい．