RT コンポーネントはどのように作ればよいか？

産業技術総合研究所 安藤慶昭, ○栗原眞二, ビグズ ジェフ, 神徳徹雄

How RT-Components should be developed?

*Noriaki ANDO, Shinji KURIHARA, Geoffrey BIGGS, Tetsuo KOTOKU, AIST

Abstract—
With the spread of RT-Middleware, various systems are being built on it. However, these systems have diverse styles, and the system developers would sometimes hesitate what kind of architecture should be taken for the target systems. In this paper, we would like to show some guide lines for RT-Middleware based robotic system development.

Key Words: RT-Middleware, RT-Component, modularization

1. はじめに

OpenRTM-aist の広がりとともに，様々なシステムが OpenRTM-aist 上に構築されつつある。ロボットシステムは，その対象や特徴は多種多様であり，同一のシステムであってもそのアーキテクチャは，開発者の経験，考え方，センスによって異なる場合が多い。

モジュール化に関する様々な研究 [1, 2] は行われているものの，コンポーネント指向によるロボットシステム開発は歴史が浅く，こうした構築手法についての明確な指針や手法が明確でない。

本稿では，著者らのこれまでの経験や，外部から OpenRTM-aist へのフィードバックをもとに，RT ミドルウエアを用いてシステムを構築する際の，コンポーネントの作成方法，システムの設計指針について示す。

2. コンポーネント開発

2·1 コールバック

RT コンポーネントは，RTC Specification によって定義されたコールバックに適切なアルゴリズムを実装することでコンポーネントを作成する。以下にコールバックを実装する際のルールを示す。

2·1.1 初期化・終了に関するルール

- 可能な限り RAII (Resource Aquisition Is Initialization) のイディオムに則り，コールバック中でのリソースの確保 (オブジェクトを new する等) を行わない．
- コールバックで動的リソースの確保を行う場合は，確保を onInitialize, 解放を onFinalize をできるだけ対にして行う．
- それ以上の動的リソース確保として onActivated, onDeactivated があるが，その場合は onAborting でのリソースの解放も行うべきである．
- 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．

2·1.2 実行に関するルール

- 通常のコンポーネント (Dataflow 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 のロジック実行時間が可変かつ (コンピュータの実行周期より十分) 長い場合には，コンポーネントアクティビティ型 (Component's activity type) を SPORADIC にしなければならない．
- 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．(OpenRTM-aist のサンプルの ConsoleIn コンポーネントは実際にはよくない例である．)

2·1.3 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 において回復不能なエラーに陥った場合は onError を対応させて実装する．
- 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．

2·2 データポート

以下に，データポートを利用する際に注意すべき点を列挙する．

- 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．
- 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．

2·3 サービスポート

以下に，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を挙げる．

- 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 内で定義されるインターフェース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．または，コンポーネントアクティビティ型を SPORADIC にする．
- 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．

2·4 コンフィギュレーション

以下に，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を挙げる．

- 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 dboule，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．
- 配列・行列についても，適切な返還関数を定義し，(C++の場合) bindParameter 関数の第 4 引数にこれを与える．

3. システム開発

システム作る際に，システム全体をどのような粒度でモジュール分割するか，モジュール間の依存性をどのように整理するかといった点に注意して全体を設計する必要がある．ここではシステムを設計・開発する際に注意ずべき点を列挙する．

3·1 モジュール分割

- 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．
- 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する．
- 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．

3·2 密結合システム

- 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．

4. その他

- 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．
- 一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．
- 言語，コンパイラ，ライブラリ，ツールの使い方に習熟し，効率のよい開発を心がける．常に「もっとよい方法があるのではないか」と考えることが大切である．
- 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．

5. おわりに

RT コンポーネント自体は，単なる分散コンポーネントのプラットフォームであるので，その利用方法は自由であるが，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．そこで本稿では，RT コンポーネントやシステムを作る際に，念頭に置くべき事項やルール等を例示した．

ここに示したルールがすべてではなく，このほかにも多くのルールが存在すると思われる．また，こうしたルールは，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．

参考文献

[1] 青木 昌彦, 安藤 晴彦, “モジュール化—新しい産業アーキテクチャの本質”, 東洋経済新報社, 2002

[2] キム・クラーク, カーリス・ボールドウィン, 安藤 晴彦 (訳), “デザイン・ルール—モジュール化パワー”, 東洋経済新報社, 2004