# RTC:Stage

Geoffrey Biggs (ジェフ・ビグズ)
geoffrey.biggs@aist.go.jp

November 5, 2010

## 1 はじめに

RTC:StageはOpenRTM-aist用のRTコンポーネントです。Stage[1]というシミュレータの仮想空間へのアクセスを提供します。仮想空間のモデルは他のコンポーネントで操作することが可能で、仮想空間内のデータを他のコンポーネントで使うことも可能です。大きな仮想空間でもコンポーネントを使いやすい小さいサイズにフィルタリングする機能も提供しています。また、コンポーネントの生成や追加によりシミュレータ側で新しいモデルを作成する事が可能なプラグイン機能もサポートしてます。

本コンポーネントの特徴の一つは、コンポーネントのポートが自動的に仮想空間に対応づけられることです。仮想空間に複数のロボットがあっても、コンポーネント側で別々のポートにより操作することが可能です。また、ポートの名前も設定することが可能です。セクション5.1を参照してください。

このソフトウェアは産業技術総合研究所で開発されています。承認番号はH22PRO-1194です。開発は新エネルギー・産業技術総合開発機構（Project for Strategic Development of Advanced Robotics Elemental Technologies）に支えられました。

## 2 条件

RTC:StageはOpenRTM-aistのC++版と最新のStage（Git リポジトリー[2]から）が必要です。Stageの最新リリースバージョンにはRTC:Stageが必要なAPIはまだ入っていません。

RTC:StageはCMake[3]を使います。Cmake 2.6以上が必要です。

プラグインを使う場合はGNUのlibtool[4]にあるlibltdlが必要です。

Stage自体はWindowsでは使えないので、コンポーネントはLinuxもしくは、MacOS Xが必要です。

## 3 インストール

パッケージを使う場合はそのパッケージの説明に従ってインストールを行ってください。

ソースからインストールを行う場合は以下の手順で進めてください。

1. ソースをダウンロードして解凍してください。

   ```
   tar -xvzf rtcstage-1.0.0.tar.gz
   ```

2. 解凍されたフォルダに入ってください。

   ```
   cd rtcstage-1.0.0
   ```

[1] http://playerstage.sourceforge.net/
[2] http://github.com/rtv/Stage
[3] http://www.cmake.org/
[4] http://www.gnu.org/software/libtool/

3. “build”フォルダを作ってください。

   ```
   mkdir build
   ```

4. “build”フォルダに入ってください。

   ```
   cd build
   ```

5. CMakeを実行してください。

   ```
   cmake ../
   ```

6. エラーが出なければ、makeを実行してください。

   ```
   make
   ```

7. make installでコンポーネントをインストールしてください。選択されたインストール場所への権限があるかを事前に確認してください。

   ```
   make install
   ```

8. インストールする場所はccmakeを実行して`CMAKE_INSTALL_PREFIX`を設定することにより変更することが可能です。

   ```
   ccmake ../
   ```

ここまでで、コンポーネントを使えるようになりました。コンフィグレーションは次のセクションを参照してください。

RTC:Stageは`rtcstage_standalone`の実行（`${prefix}/bin`にインストールされてます）によりスタンドアローンモードで実行することができます。あるいは、`librtcstage.so`を初期化関数の`rtcstage_init`を利用して、マネージャにロードすることも可能です。このライブラリは`${prefix}/lib` または`${prefix}/lib64`にインストールされてます。

# 4 コンフィグレーション

RTC:Stageは仮想空間と一致するポートをダイナミックに作るため特別な起動方法を使います。このため、コンフィグレーションはRTSystemEditorやrtshellによってではなく、rtc.confなどのコンフィグレーションファイルでパラメータを設定する必要があります。

コンポーネントのコンフィグレーションを設定するために、以下のようなファイルを作ってください。

```
configuration.active_config: simple

conf.simple.world_file: /usr/local/share/stage/worlds/simple.world
conf.simple.gui_x: 640
conf.simple.gui_y: 480
conf.simple.limit_models:
```

一つのファイルで複数のコンフィグレーションセットを設定することができます。上記ファイルの最初の行では、起動時のコンフィグレーションセットを指定してます。ファイルに正しい名前をつけ（例：“stage.conf”）、`rtc.conf`に以下の行を追加してください。

```
Simulation.RTC_Stage.config_file: stage.conf
```

本コンポーネントで使えるコンフィグレーションパラメータについては、テーブル 1を参照してください。

# 5 ポート

コンポーネントは初期化される時に提供するポートをダイナミックに作ります。

| パラメータ | 意味 |
|---|---|
| world_file | ロードするワールドファイルです。Stageの説明書を参照してください。Stageをインストールする時、サンプルワールドファイルがインストールされます。 |
| gui_x | シミュレータのウィンドウの幅です。 |
| gui_y | シミュレータのウィンドウの高さです。 |
| limit_models | モデルのフィルターです。セクション 6.1を参照してください。 |
| plugins | プロキシープラグインです。セクション 6.2を参照してください。 |

Table 1: コンフィグレーションパラメータ。

| モデル種類 | モデル名 | ポート名 |
|---|---|---|
| Robot | r0 | r0_vel_control |
| Laser | r0.laser:0 | r0_laser_0_ranges |
| Camera | r0.camera:1 | r0_camera_1_image |

Table 2: ポート名の例。

### 5.1 ポート名

ポート名は仮想空間を反映し、それらがどのモデルにアクセスを提供するかを示します。例えば、仮想空間に「r0」という名前のロボットがあれば、そのロボットの速度コントロール、オドメトリ出力、幾何学サービスなどへのアクセスを提供するポートが作られます。これらのポートはすべて接頭辞に「r0_」が付きます。ポートがどのように作成されるかの例については、テーブル 2を参照してください。特殊文事（「.」と「:」）が「_」に取り替えられることに注意してください。

## 6 モデルプロキシー

RTC:Stageコンポーネントは、仮想空間に含まれているモデルへのアクセスを提供するためにモデルプロキシーを使用します。仮想空間のモデルのインスタンスはそれぞれコンポーネントのモデルプロキシーのインスタンスに直接対応付けられます。いくつかのプロキシーはコンポーネントによって提供されています。これらはStageでサポートされている最も人気なモデルをカバーします。これらはテーブル 3で述べられています。

プロキシーが提供されていないStageのモデルを使用したい場合、プラグインプロキシーを作ることが可能です。詳細についてはセクション 6.2を参照してください。

### 6.1 モデルフィルタ

多くのモデルのシミュレーションを行う場合、プロキシーされたモデル、そしてコンポーネントによって提供されるポートの数は収集不可能になるかもしれません。これを回避するために、ユーザはコンポーネントのコンフィグレーションでモデル名フィルタを指定することができます。コンポーネントはフィルタに含まれたモデルのプロキシーだけを作成します。

フィルタのフォーマットはコンマによって区切られたストリングのリストです。それぞれのストリングがフィルタです。モデル名は、プロキシーが作成されるために少なくとも一つのフィルタと一致する必要があります。ワイルドカード（「*」）はフレキシブルなフィルタを指定するために使用することができます。フィルタフォーマットをテーブル 4に示します。

例えば、2台のロボット、「r0」及び「r1」を含んでいるシミュレーションを考慮してください。「r0」にはレーザーセンサー及びカメラがあります。「r1」には二つのレーザーセンサーがあります。シミュレートされたコンポーネントは、以下のモデルにプロキシーを提供します。

| プロキシー | ポート | データ形 | ポートの意味 |
|---|---|---|---|
| Actuator | vel_control | TimedDouble | アクチュエータの速度制御。 |
| | pos_control | TimedDouble | アクチュエータの位置制御。 |
| | state | ActArrayState | アクチュエータの現在の状態。 |
| | current_vel | TimedDouble | アクチュエータの現在の速度。 |
| | svc | GetGeometry2D | アクチュエータの位置とサイズの取得。 |
| Camera | control | TimedPoint2D | パンとティルトに対するコントロール。 |
| | image | CameraImage | カメラからのイメージ（RGBA）。 |
| | depth | CameraImage | カメラからの深さイメージ（8ビット）。 |
| | svc | GetGeometry2D | カメラの位置とサイズの取得。 |
| Fiducial | fiducials | Fiducials | 現在検知されたfiducialのリスト。 |
| | svc | GetGeometry2D | Fiducialセンサーの位置とサイズの取得。 |
| Gripper | state | GripperState | グリッパーの状態。 |
| | svc | GetGeometry2D | グリッパーの位置とサイズの取得。 |
| | | GripperControl | グリッパーの開閉。 |
| Laser | ranges | RangeData | レンジデータ。 |
| | intensities | IntensityData | インテンシティーデータ |
| | svc | GetGeometry2D | レーザの位置とサイズの取得。 |
| Position | vel_control | TimedVelocity2D | ロボットの速度制御。 |
| | pose_control | TimedPose2D | ロボットの位置制御。 |
| | current_vel | TimedVelocity2D | ロボットの現在の速度。 |
| | odometry | TimedPose2D | ロボットの現在のオドメトリー。 |
| | svc | GetGeometry2D | ロボットの位置とサイズの取得。 |
| | | SetOdometry2D | ロボットの現在のオドメトリーの設定。 |

Table 3: RTC:Stageのプロキシー。

- r0
- r0.camera:0
- r0.laser:0
- r1
- r1.laser:0
- r1.laser:1

フィルタなしでは、これは多くのポートを備えたRTC:Stageのインスタンスを生産します。ユーザが単に利用可能なモデルの一部だけに興味を持っていれば、適切なフィルタによってプロキシーの数を制限することができます。テーブル 5は、異なるフィルタストリングでプロキシーされたモデルの例を示します。

## 6.2 プラグインプロキシー

Stageシミュレータはモデルプラグインを書くことをサポートします。これらは、新しいデバイスタイプがStage自体を修正せずに、容易にシミュレートすることを可能にするため、シミュレーションでの機能の追加を提供します。多くのロボット開発者がこの方法で新しい装備を実装したいと思うかもしれません。そのようなモデルはRTC:Stageではデフォルトではサポートされません。サポートをするために、モデルプラグインに合うプロキシープラグインを作成しなければなりません。プロキシープラグインも`ModelPosition` モデルのようなStageに組み込まれたモデルのために作成することができます。また、RTC:Stageに含まれたプロキシーを無視する、新しいプロキシーも作成することができます。

| フィルター | 影響 |
|---|---|
| `filter` | モデル名が全て一致。 |
| `*filter` | モデル名の最後のストリングが一致。 |
| `filter*` | モデル名の先頭のストリングが一致。 |
| `*filter*` | モデル名の一部のストリングが一致。 |
| `filter1*,filter2*` | 二つのフィルター。 |

Table 4: フィルタフォーマットの例。

| フィルタ | 作成されたプロキシー |
|---|---|
| `r0` | r0 |
| `r0*` | r0, r0.camera:0, r0.laser:0 |
| `*camera:0` | r0.camera:0 |
| `*:0` | r0.camera:0, r0.laser:0, r1.laser:0 |
| `*laser*` | r0.laser:0, r1.laser:0, r1.laser:1 |
| `r0.laser*,r1.laser*` | r0.laser:0, r1.laser:0, r1.laser:1 |
| `r1,*laser:0` | r0.laser:0, r1, r1.laser:0, r1.laser:1 |

Table 5: いろいろなフィルタの結果で作成されたプロキシー。

プロキシープラグインは`ModelProxy`インターフィースのインプリメンテーションを提供します。それは、このインターフェースの抽象メソッドを実装しなければならず、RTC:Stageコンポーネントにポートを関連付けるために重要です。されに、それはこれらのポートとシミュレーションの間のデータをやり取りするために重要です。

されに、プロキシープラグインは2つのシンボルを書き出す必要があります：

- `GetProxyType` - プラグインのモデル種類を返す。
- `ProxyFactory` - プラグインのインスタンスを作成する.

プロキシープラグインをコンパイルするには`BUILD_PROXY_PLUGIN`というCMake用のマクロを使ってください。`RTCStagePlugin`というCMakeファイルで提供されています。

RTC:Stageにはプラグインのサンプルがあります。`${prefix}/share/rtcstage/examples/`にインストールされます。プロキシープラグインの作成についての詳細については、これらのサンプルを参照してください。一般に、サンプルをコピーしソースを新しいモデルに合わせる用に修正する方が開発が早いでしょう。

## 6.3 プラグインの例

RTC:Stageには二つのプラグインのサンプルがあります。`${prefix}/share/rtcstage/examples/`にインストールされます。`${prefix}`はRTC:Stageがインストールされたフォルダです。プラグインはCMakeでコンパイルすることができます。

1. `cd ${prefix}/share/rtc\_stage/examples/blobfinder/`
2. `mkdir build`
3. `cd build`
4. `cmake ../`
5. `make`

### 6.3.1 Blobfinderプロキシー

このサンプルプラグインはStageのblobfinderセンサーモデルにプロキシーを提供します。必要なデータ型を提供するためにプラグインでユーザのIDLファイルを使用するサンプルを示してます。

### 6.3.2 Positionプロキシー

このプラグインはデフォルトの位置モデルプロキシーをカスタムプロキシーに取り替えて明示してます。このプラグインによって提供されるプロキシーを読み込む事によって、デフォルトのpositionプロキシーはコンポーネント中で無視されます。新しいプロキシーは異なるデータ型を使用するため、モデルに代替インターフェースを提供します。