次世代ロボット知能化技術開発プロジェクト

施設内生活支援ロボット知能の研究開発

作業計画モジュール(SDLEngine)

マニュアル

Ver. 4.0

2012 年 1 月

九州工業大学

改変履歴

Ver.	改版日	改変内容
1.0	2010/1/12	新規作成
2.0	2010/3/10	データポートサポート版
2.1	2010/11/10	ドキュメントの訂正
2.2	2011/1/15	モジュールとドキュメントの不整合を対応
3.0	2011/1/22	OpenRTM-aist 1.0.0 に対応
3.1	2011/7/6	ドキュメントのミスの訂正
3.2	2011/11/23	ドキュメントのミスの訂正
3.3	2011/11/25	JDK1.7.0 に対応、eclipse のインポート方法の変更に対応、ファイル名変更、標準のポートの説明
3.4	2011/12/7	ドキュメントの補足の追加
3.5	2011/12/14	SpeechDetector 等の公開への対応
3.6	2011/12/27	ドキュメントのミスの訂正
3.7	2012/1/4	ドキュメントのミスの訂正
4.0	2012/1/17	コンソールなしの実行に対応、ネームサーバのポート変更方法の追加

1. はじめに

1.1. 本文書について

本書は、「次世代ロボット知能化技術開発プロジェクト」の「施設内生活支援ロボット知能研究開発」において構築した作業計画モジュールについてのマニュアルです。本書は RT ミドルウェア（以下 RTM）、RT コンポーネント（以下 RTC）を用いたロボットシステム開発者を対象に記述されており、RTM、RTC や関連ツールに関する一般的な知識を持つことを前提とします。

OpenRTM-aist official webside: http://www.openrtm.org/openrtm/ja/

1.2. 対象プラットホーム

本モジュールは、以下の環境で動作確認しています。

- Windows XP professional 32bit
- Windows Vista Business 32bit/64bit
- Windows 7 Professional/Enterprise 32bit/64bit
- Ubuntu Linux

※ 以下の説明は Windows 7 64bit 版を想定して記述しています。他の環境については適宜書き換えて対応ください。

1.3. OpenRTM-aist-C++/Python 関係のインストール

以下のソフトウェアは、本モジュールでは使用していないがインストールしていることが望ましいものです。

- Microsoft Visual Studio 2008 Express/Professional Edition
- OpenRTM-aist-1.0.0-RELEASE_vc9_100212.msi
- OpenCV_1.0.exe
- Python-2.6.4.msi
- PyYAML-3.09.wi32-py2.6.exe

※ Linux の場合は、pkg_install_XX.sh (XX: vine, fedra, Ubuntu, debian) でインストールしてください。

1.4. OpenRTM-aist-Java 関係のインストール

以下のソフトウェアをインストールしてください。

- Java SE Development Kit JDK 1.6.0_XX（Eclipse の関係で 32bit 版が必要）
  - 環境変数 JAVA_HOME に C:¥Program Files¥Java¥jdk1.6.0_XX を設定する。
  - 64bit 環境に 32bit 版をインストールする場合は以下の通りにする。
  - 環境変数 JAVA_HOME に C:¥Program Files (x86)¥Java¥jdk1.6.0_XX を設定する。

- 環境変数 PATH に %JAVA_HOME%¥bin; を追加する。

- OpenRTM-aist-Java-1.0.0-RELEASE
  - OpenRTM-aist-Java-1.0.0.msi を実行する。
  - 環境変数 RTM_JAVA_ROOT に C:¥Program Files (x86)¥OpenRTM-aist¥1.0 を設定する。
- Eclipse 3.4.2 (Ganymede SR2)
  - OpenRTM 全部入り eclipse342_rtmtools110-rc2_wn32_ja.zip 等をインストールする。
  - 或は eclipse と rtmtools110-rc2_ja.zip 等をインストールする。

  ※ eclipse が 32bit 版なので、JDK は 32bit 版が必要です。

  ※ eclipse の「ウィンドウ」メニューの「設定」の 「Java」の「コンパイラ」の「コンパイラ準拠レベル」を 1.5 に設定する。
- Apache Ant 1.8.2
  - 環境変数 ANT_HOME に C:¥apache-ant-1.8.2 を設定。
  - 環境変数 PATH に %ANT_HOME%¥bin; を追加する。

  ※ eclipse がインストールされている場合は、環境変数 ANT_HOME に $ECLIPSE_HOME/plugins/org.apache.ant_xxx を設定して利用してもよい。

2. 作業計画モジュール

SDLEngine は、様々な RTC と接続して作業計画のアプリケーションを実行するためのモジュールです。Java のインタプリタである BeanShell を組み込んでおり、Java 及びそのインタプリタであるスクリプトを用いてアプリケーションを記述できます。電気通信大学の末廣先生作成の RtcHandle と同じようなことができます。

2.1. インストール

RTC 再利用センター登録の zip ファイルをダウンロードしてください。

作業計画モジュール: SDLEngine.4.0.zip

以下の手順で eclipse にプロジェクトをインポートします。

1. eclipse を起動する。
2. Java パースペクティブにする。
3. ファイルメニューから「インポート」を選択し、「一般」の「既存のプロジェクトをワークスペースへ」を選ぶ。
4. 次に進み、「アーカイブ・ファイルの選択」の欄にチェックし、「参照」で SDLEngine.4.0.zip を選択する。
5. 「完了」を選択する。
6. インポートした段階では、いくつかのディレクトリが不足していてエラーが出ている。そこで、SDLEngine 以下に lib と rtc/java のディレクトリを作成する。

2.2. SDLEngine のビルド

1. ビルドには、Java と OpenRTM を除き、特に必要なソフトウェアはありません。
2. ビルドに関しては次の方法があります。
   - Eclipse によるビルドを実行して下さい。
     また、「自動的にビルド」が有効になっていれば、特に行うことはありません。
     ビルド後に問題が発生していなければ、ビルドは完了です。
   - Ant を使用してビルドを行います。
     ビルドファイルである SDLEngine/build.xml 中の compile ターゲットを起動します。
3. エラーが出る場合は、以下の点を確認してください。
   - コンパイラ準拠レベルが 1.5 に設定されているか。
   - JAVA_HOME にインストールされている JDK が正しく設定されているか。
   - JAVA_HOME/bin にパスが設定されているか。
   - SDLEngine/build.xml 中の clean ターゲットを実行してから再度ビルドしてみる。
   - SDLEngine/lib と SDLEngine/rtc/java が存在しているか。

2.3. SDLEngine の起動

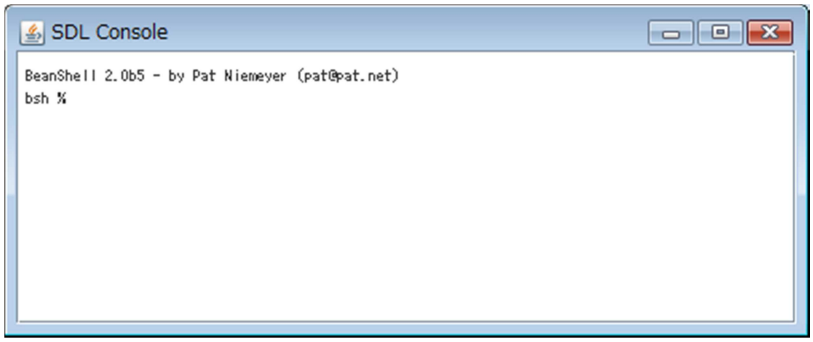
起動時の設定ファイルである SDLEngine/rtc.conf の内容を確認して下さい。

- corba.nameservers: CORBA ネームサーバのホスト名とポート番号
  例: corba.nameservers: localhost:5005
- ホスト名とポート番号を変更するときは、一度 clean してください。

次の何れかの方法で SDLEngine を起動します。

- SDLEngine/build.xml 中の run-console ターゲットを起動します
- Eclipse より jp.ac.kyutech.SRP.Console クラスを起動します
- Ant を利用した bat ファイル(SDLEngine/doc/SDLEngine.bat)を起動します

正常に起動すると SDLEngine のコンソールウィンドウが表示されます。



注意事項

- idlj が見つからない などのエラーが出た場合は、JAVA_HOME に JDK のディレクトリが設定されており、その下の bin へパスが通っていることを確認して下さい。
- Eclipse のデフォルトの Java VM を JDK にしておいてください。

2.4. SDLEngine スクリプト

SDLEngine スクリプトは、BeanShell を基盤として利用しています。従って BeanShell の機能は全て有しています。詳しくは、SRPCommon/lib/beanshell/bshmanual.pdf を参照してください。

主な機能は次の点です。

- Java 1.4 に準拠したインタプリタ
- 既存の Java クラスのインスタンス化と実行
- BeanShell で用意された関数群の利用

さらに、コンソールの起動時に、SDLEngine 用のクラスやユーティリティを読込んでいます。これは、SDLEngine/src/jp/ac/kyutech/SRP/Scripting/Scripts/setup.bsh から、rtc.bsh と util.bsh を読込むことで実現しています。次に重要なオブジェクトを説明します。

(1) rtc.env オブジェクト

SDLEngine で RTC オブジェクト群を操作するために、必要な情報を保持し、便利なメソッドを提供する Bsh Object です。

- コンストラクタ: Corba Name Server のホスト名とポート番号を引数で指定します。
- get_names メソッド: Corba Name Server に接続されている全オブジェクト名を取得します。
- get_handles メソッド: get_names()で取得される名前リストに対して rtc.handle でラッピングします。ただし、RTC オブジェクトで無い Corba オブジェクトに関しては失敗します。(無視して問題ありません)
  以降の connect メソッドや、handles 変数を扱う前に呼び出してください。また途中でコンポーネントが追加される場合にも呼び出す必要があります。
- connect メソッド: 引数で指定された 2 ポートを接続します。
- handles 変数: Corba Name Server 上での RTC の名前をキーに rtc.handle オブジェクトを保持します。この名前は、RtcLink、RTSystemEditor などで表示されるコンポーネントのインスタンス名と同じになります。
  例: SDLEngine コンポーネントを初期化して、活性化します。

```
env = rtc.env("localhost", 5005);
sdlEngine = rtc.local_component("SDLEngine", "SDLEngine");
env.get_handles();
env.handles["SDLEngine0.rtc"].activate();
```

(2) rtc.local_component オブジェクト

SDLEngine と同じ JVM 上(ローカル)で動作する RTC オブジェクトを操作するために、ラッピングする Bsh Object です。RTC オブジェクトと一対一です。

- get_ports メソッド: RTC における RTC.Port クラスの配列を取得します。
- local_ports 変数: ポートを操作するためのオブジェクトを、名前をキーにして保持します。
  これにより、各ポートに対応する RTC の Java メソッドを SDLEngine スクリプトから直接起動できます。
  例: SDLEngine コンポーネントの myservice ポートに対応する RTC オブジェクトの echo メソッドを起動します。

```
sdlEngine.local_ports["myervice"].echo("hello world");
```

(3) ユーティリティメソッド

- encode64 メソッド: 引数で指定された文字列を Base64 でエンコードします。
- decode64 メソッド: 引数で指定された文字列を Base64 でデコードし、文字列として返します。データの中身が文字データであることを前提としています。
- match メソッド: 第 1 引数の文字列と第 2 引数の正規表現をマッチさせます。マッチする場合は true を返します。結果は、$match と$m にて参照できます。
- encodexml メソッド: 指定されたオブジェクトを XML 形式に変換します。
- decodexml メソッド: 指定された XML 表記の文字列をオブジェクトに変換します。

その他詳細は、スクリプトのソースファイルを参照して下さい。

2.5. デフォルトの SDLEngine のポートについて

SDLEngine では、build.xml の記述によりサービスポートやデータポートを自由に追加や削除を行えます。ここでは、インストールしたデフォルトの状態の build.xml とポートの構成を紹介します。

build.xml の gen-rtc-sdl-engine ターゲット

```
<target name="gen-rtc-sdl-engine" description="Generate SDL Engine RTC code.">
    <java classname="jp.go.aist.rtm.rtctemplate.CuiRtcTemplate" fork="true" failonerror="true">
        <classpath refid="libs.path" />
        <classpath refid="libs.rtctemplate.sdl.path" />
        <arg value="--output=${rtc.java.dir}" />
        <arg value="--backend=java" />
        <arg value="--module-name=SDLEngine" />
        <arg value="--module-desc='SDL Engine'" />
        <arg value="--module-vender='Kyushu Institute of Technology'" />
        <arg value="--module-category=Both" />
        <arg value="--module-comp-type=DataFlowComponent" />
        <arg value="--module-act-type=SPORADIC" />
        <arg value="--module-max-inst=1" />
        <arg value="--consumer=MyServiceConsumer:myservice:jp::ac::kyutech::SRP::demo::MyService" />
        <arg value="--consumer-idl=${main.idl.dir}/MyService.idl" />
        <arg value="--service=SpeechDetecter:speechDetecter:SpeechDetecter" />
        <arg value="--service-idl=${main.idl.dir}/SpeechDetecter.idl" />
        <arg value="--idl-include=${main.idl.dir}" />
        <arg value="--inport=in:RTC::TimedLong" />
        <arg value="--outport=out:RTC::TimedLong" />
        <arg value="--inport=myDataIn:jp::ac::kyutech::SRP::demo::MyData" />
        <arg value="--outport=myDataOut:jp::ac::kyutech::SRP::demo::MyData" />
        <arg value="--inport=stringIn:RTC::TimedString" />
        <arg value="--outport=stringOut:RTC::TimedString" />
    </java>
</target>
```

サービスポートとデータポートに関する定義部分を説明します。

サービスポート(コンスーマ)の定義部分

```
<arg value="--consumer=MyServiceConsumer:myservice:jp::ac::kyutech::SRP::demo::MyService" />
<arg value="--consumer-idl=${main.idl.dir}/MyService.idl" />
```

MyService.idl で定義したサービスポートをコンスーマとして作成する記述です。

ここで、MyService.idl は以下の通りです。

```
module jp {
  module ac {
    module kyutech {
      module SRP {
        module demo {
          typedef sequence<string> EchoList;
          typedef sequence<float> ValueList;
          interface MyService
          {
            string echo(in string msg);
            EchoList get_echo_history();
            void set_value(in float value);
            float get_value();
            ValueList get_value_history();
          };

        };
      };
    };
  };
};
```

発話推定モジュール(SpeechDetector)(九工大)を接続するためのサービスポートの定義部分

```
<arg value="--service=SpeechDetecter:speechDetecter:SpeechDetecter" />
<arg value="--service-idl=${main.idl.dir}/SpeechDetecter.idl" />
```

データポートの定義

```
<arg value="--idl-include=${main.idl.dir}" />
<arg value="--inport=in:RTC::TimedLong" />
<arg value="--outport=out:RTC::TimedLong" />
<arg value="--inport=myDataIn:jp::ac::kyutech::SRP::demo::MyData" />
<arg value="--outport=myDataOut:jp::ac::kyutech::SRP::demo::MyData" />
<arg value="--inport=stringIn:RTC::TimedString" />
<arg value="--outport=stringOut:RTC::TimedString" />
```

データポート等で使用する idl を保存するディレクトリの指定し、そこに含まれる idl ファイルを読み込みます。ここで定義しているデータポートは以下の通りです。

定義しているポートの一覧

ポートタイプ	ポート名	ポートのデータ型	説明
inport	in	RTC::TimedLong	ConsoleInComp 等用
outport	out	RTC::TimedLong	ConsoleOutComp 等用
inport	myDataIn	jp::ac::kyutech::SRP::demo::MyData	独自データポート説明用
outport	myDataOut	jp::ac::kyutech::SRP::demo::MyData	独自データポート説明用
inport	stringIn	RTC::TimedString	音声認識等用
outport	stringOut	RTC::TimedString	音声合成等用

build.xml では、SDL Engine 以外のダミーモジュールを作成し Bsh のコンソールと接続することができます。標準で用意されているダミーモジュールを紹介します。

MyService Blank Provider RTC は、SDLEngine と接続した動作確認を行うためのダミーモジュールです。次の章でデモを紹介します。

MyService 型のサービスポートを持つダミーモジュールの生成部分

```
<target name="gen-rtc-myservice-blank-provider" description="Generate MyService Blank Provider RTC code.">
  <java classname="jp.go.aist.rtm.rtctemplate.CuiRtcTemplate" fork="true" failonerror="true">
    <classpath refid="libs.path" />
    <classpath refid="libs.rtctemplate.sdl.path" />
    <arg value="--output=${rtc.java.dir}" />
    <arg value="--backend=java" />
    <arg value="--module-name=MyServiceBlankProvider" />
    <arg value="--module-desc='MyService Blank Provider'" />
```

```
    <arg value="--module-vender='Kyushu Institute of Technology'" />
    <arg value="--module-category=example" />
    <arg value="--module-comp-type=DataFlowComponent" />
    <arg value="--module-act-type=SPORADIC" />
    <arg value="--module-max-inst=1" />
    <arg value="--service=MyServiceBlankProvider:myservice:jp::ac::kyutech::SRP::demo::MyService" />
    <arg value="--service-idl=${main.idl.dir}/MyService.idl" />
  </java>
</target>
```

次に、RTC::TimedLong 型の出力データポートを持つダミーモジュールと RTC::TimedLong 型の入力データポートを持つダミーモジュールを紹介します。これらは、ConsoleIn と ConsoleOut と同じように使用することができます。

RTC::TimedLong 型の出力データポートを持つダミーモジュールの生成部分

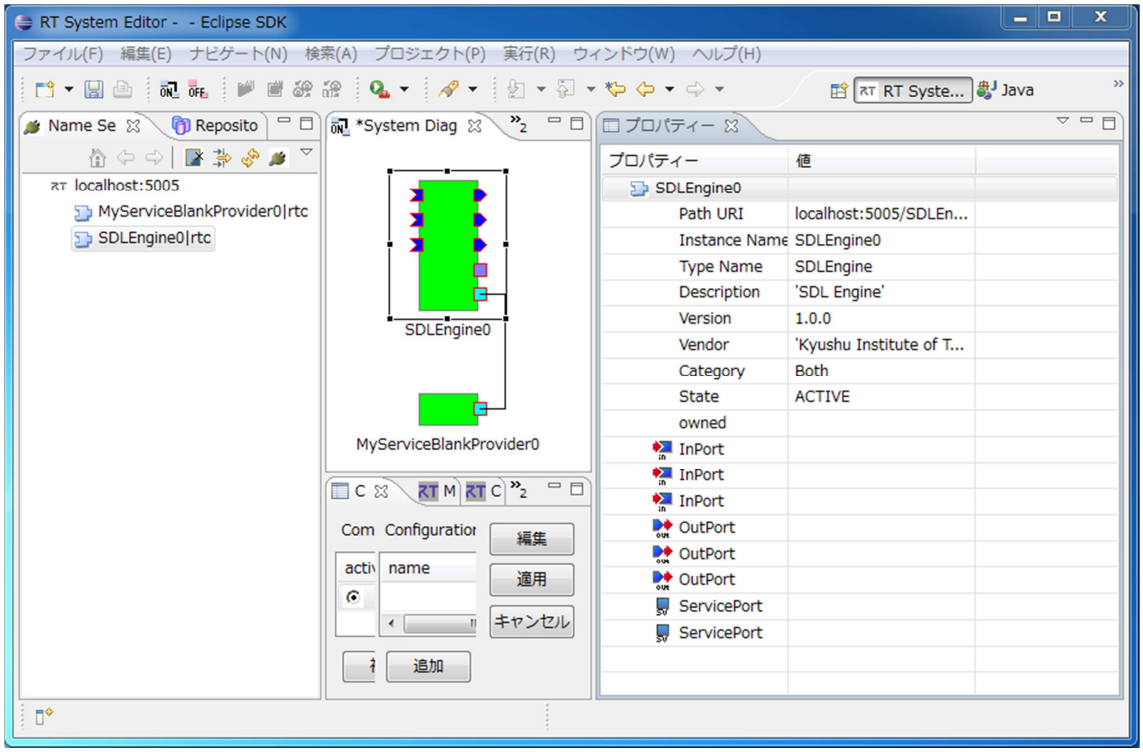
```
<target name="gen-rtc-sdl-console-in">
  <java classname="jp.go.aist.rtm.rtctemplate.CuiRtcTemplate" fork="true" failonerror="true">
    <classpath refid="libs.path" />
    <classpath refid="libs.rtctemplate.sdl.path" />
    <arg value="--output=${rtc.java.dir}" />
    <arg value="--backend=java" />
    <arg value="--module-name=SDLConsoleIn" />
    <arg value="--module-desc='SDL Console input component'" />
    <arg value="--module-vender='Kyushu Institute of Technology'" />
    <arg value="--module-category=example" />
    <arg value="--module-comp-type=DataFlowComponent" />
    <arg value="--module-act-type=SPORADIC" />
    <arg value="--module-max-inst=10" />
    <arg value="--outport=out:RTC::TimedLong" />
   </java>
</target>
```

RTC::TimedLong 型の入力データポートを持つダミーモジュールの生成部分

```
<target name="gen-rtc-sdl-console-out">
  <java classname="jp.go.aist.rtm.rtctemplate.CuiRtcTemplate" fork="true" failonerror="true">
    <classpath refid="libs.path" />
```

```
    <classpath refid="libs.rtctemplate.sdl.path" />
    <arg value="--output=${rtc.java.dir}" />
    <arg value="--backend=java" />
    <arg value="--module-name=SDLConsoleOut" />
    <arg value="--module-desc='SDL Console output component'" />
    <arg value="--module-vender='Kyushu Institute of Technology'" />
    <arg value="--module-category=example" />
    <arg value="--module-comp-type=DataFlowComponent" />
    <arg value="--module-act-type=SPORADIC" />
    <arg value="--module-max-inst=10" />
    <arg value="--inport=in:RTC::TimedLong" />
  </java>
</target>
```

SDLEngine と MyService Blank Provider RTC を接続した状態の RT System Editor



2.6. SDLEngine のサービスポートのデモ

新たにサービスポートを組み込む方法は、後で説明します。ここでは、組み込まれている状況でデモを実行します。

(1) 前準備

1. ネームサーバを起動します。

   以下の方法を参考にネームサーバを起動してください。

   (ア) Eclipse の SDLEngine プロジェクト内の OmniNames.bat を実行する。

   ※ 環境変数 OMNI_ROOT が設定されていることを確認してください。

   ※ 必要に応じて OmniNames.bat を書き換えてください。

2. 作業計画モジュールのコンソールを起動

   以下のいずれかの方向で SDLEngine のコンソールを起動します。

   (ア) Eclipse の SDLEngine プロジェクト内にある
   jp.ac.kyutech.SRP.Console クラス
   を Java アプリケーションとして起動

   (イ) Eclipse の SDLEngine プロジェクト内の build.xml の run-console のターゲットを実行

   (ウ) Eclipse の SDLEngine プロジェクト内の doc 中の SDLEngine.bat を実行

   ※ 必要に応じて SDLEngine.bat を書き換えてください。

(2) デモ

1. スクリプトの投入

行動計画コンソールに以下のスクリプトをコピペして enter を入力して実行して下さい。

```
// name サーバの設定
env = rtc.env("localhost", 5005);
// MyServiceBlankProvider をローカルオブジェクトとして起動
myServiceProvider = rtc.local_component("MyServiceBlankProvider", "MyServiceBlankProvider");
// SDLEngine をローカルオブジェクトとして起動
sdlEngine = rtc.local_component("SDLEngine", "SDLEngine");
// name サーバに登録されているオブジェクトのハンドラを取得
env.get_handles();
// myServiceBlankProvider0.rtc のポートと SDLEngine0.rtc のポートを接続
env.connect(env.handles["MyServiceBlankProvider0.rtc"].ports["MyServiceBlankProvider0.MyServiceBlankProvider"],
env.handles["SDLEngine0.rtc"].ports["SDLEngine0.MyServiceConsumer"]);
// myServiceBlankProvider0.rtc を活性化
env.handles["MyServiceBlankProvider0.rtc"].activate();
// SDLEngine0.rtc を活性化
```

```
env.handles["SDLEngine0.rtc"].activate();
// myServiceProvider に対し、myservice に対応するリスナーを設定（実装に相当）
myServiceProvider.local_ports["myservice"].addListener(new jp.ac.kyutech.SRP.Scripting.ProviderListener() {
  invoke(name, args) {
    print("Method: " + name + " invoked: " + java.util.Arrays.asList(args));
  }
});
// SDLEngine の myservie にある echo("hello world")を起動
print(sdlEngine.local_ports["myservice"].echo("hello world"));
```

2. 以下のような表示が見えたら成功です。

```
Method: echo invoked: [hello world]
```

3. 説明

このデモでは、RT コンポーネントを 2 つ作成し、コンポーネント間での通信が行えていることが確認できます。SDLEngine のから MyServiceBlankProvider に対して、echo()メソッドで "hello world" のメッセージを送信しています。そのメッセージを受信した MyServiceBlankProvider がコンソールに送られてきた文字を出力しています。

ここで SDLEngine は Consumer、MyServiceBlankProvider は Provider として振舞います。

2.7. 新しいサービスポートを SDLEngine に組み込む

新しいサービスポートに対応する IDL ファイル用意して、それを SDLEngine にサービスポートとして組み込む手順を紹介します。

1. IDL ファイルを SDLEngine/main/idl に配置します。ここでは、MyService.idl とします。
2. build.xml の gen-rtc-sdl-engine ターゲットを修正します。

- Consumer として登録する場合

```
<arg value="--consumer=MyServiceConsumer:myservice:jp::ac::kyutech::SRP::demo::MyService" />
<arg value="--consumer-idl=${main.idl.dir}/MyService.idl" />
<arg value="--idl-include=${main.idl.dir}" />
```

- Provider として登録する場合

```
<arg value="--provider=MyServiceProvider:myservice:jp::ac::kyutech::SRP::demo::MyService" />
<arg value="--provider-idl=${main.idl.dir}/MyService.idl" />
<arg value="--idl-include=${main.idl.dir}" />
```

なお <arg value > の行は OpenRTM の rtc-template のコマンドライン引数となりますので、そちらを参照して書き換えても問題ありません。

3. Ant の clean ターゲットと rtc.compile ターゲットを起動します。

以上の手順で、SDLEngine に IDL ファイルで定義されたサービスポートを組み込むことができます。

2.8. 新しい RT コンポーネントのサービスポートを SDLEngine に対応させる

ここでは、IDL ファイルから SDLEngine に対応した、空の Provider RT コンポーネントを作成する手順を紹介します。これにより、SDLngine に対応した相手方のコンポーネントのダミー版を作成し、SDLengine と同じ VM 上で動作させることができます。

1. IDL ファイルを SDLEngine/main/idl に配置します。
2. build.xml の gen-rtc-template ターゲットを元に、Ant のターゲットを追加します。
   引数の詳細は OpenRTM のマニュアルを参照して下さい。
   例: Foo サービスを追加する

```
<target name="gen-rtc-foo">
  <java classname="jp.go.aist.rtm.rtctemplate.CuiRtcTemplate" >
    <classpath refid="libs.path" />
    <classpath refid="libs.rtctemplate.sdl.path" />
    <arg value="--output=${rtc.java.dir}" />
    <arg value="--backend=java" />
    <arg value="--module-name=FooProvider" />
    <arg value="--module-desc='Foo Provider'" />
    <arg value="--module-vender='Kyushu Institute of Technology'" />
    <arg value="--module-category=Provider" />
    <arg value="--module-comp-type=DataFlowComponent" />
    <arg value="--module-act-type=SPORADIC" />
    <arg value="--module-max-inst=1" />
    <arg value="--service=FooProvider:hogeService:FooService" />
    <arg value="--service-idl=${main.idl.dir}/Foo.idl" />
  </java>
</target>
```

3. build.xml の rtc.all ターゲットに、作成した gen-rtc-* ターゲットを追加します。

```
<target name="rtc.all" description="Generate RTComponents code.">
  ....
  <!-- Append rtc generation target -->
  <antcall target="gen-rtc-sdl-engine" />
  <antcall target="gen-rtc-foo" />
  <!-- Append rtc generation target -->
  ....
</target>
```

4. Ant の clean ターゲットと rtc.compile ターゲットを起動します。

```
$ ant clean rtc.compile
```

以上の手順で、空の Provider RT Component が作成できます。このコンポーネントを用いて、SDLEngine のスクリプトの動作検証を行うことができます。

2.9. SDLEngine のデータポートのデモ1

データポートを組み込む方法は、後で説明します。ここでは、組み込まれている状況でデモを実行します。入力した値をデータポート経由で SDLEngine に送信し、コンソール上で表示するデモです。

1. build.xml の run-console ターゲットを起動して下さい。
   SDLEngine RT Component と SDLConsoleIn RT Component が作成されます。
2. 起動されたコンソールにて、次のスクリプトを動作して下さい。

```
// name サーバの設定
env = rtc.env("localhost", 5005);
// SDLEngine をローカルオブジェクトとして起動
sdlEngine = rtc.local_component("SDLEngine", "SDLEngine");
// コンソールから入力を受け付けるローカルコンポーネントを起動
sdlConsoleIn = rtc.local_component("SDLConsoleIn", "SDLConsoleIn");
// name サーバに登録されているオブジェクトのハンドラを取得
env.get_handles();
// SDLEngine0.rtc の in ポートと SDLConsoleIn0.rtc の out ポートを接続
env.connect(env.handles["SDLEngine0.rtc"].ports["SDLEngine0.in"],
    env.handles["SDLConsoleIn0.rtc"].ports["SDLConsoleIn0.out"]);
// SDLEngine0.rtc を活性化
env.handles["SDLEngine0.rtc"].activate();
// SDLConsoleIn0.rtc を活性化
env.handles["SDLConsoleIn0.rtc"].activate();
// sdlEngine の in ポートに対応するリスナーを設定（実装に相当）
sdlEngine.local_ports["SDLEngine0.in"].addListener(new jp.ac.kyutech.SRP.Scripting.InPortListener() {
  dataReceived(event) {
    value = event.getValue();
    print("Received: " + value.data + " (" + value.tm.sec + "," + value.tm.nsec + ")");
  }
});
```

3. 次のスクリプトを実行すると、SDLConsoleIn の out データポートに"1"を送信し、

```
sdlConsoleIn.local_ports["SDLConsoleIn0.out"].put(1);
```

結果として、SDLEngine の in データポートで受信し、コンソールに

```
Received: 1 (1295249314, 3400000000)
```

を表示します。

Timed 型などで時刻を指定して送信するには、次のように送信し、

```
sdlConsoleIn.local_ports["SDLConsoleIn0.out"].put(1, new RTC.Time(1234,0));
```

```
Received: 1 (1234,0)
```

を表示します。

4. データポートのイベントを対応したコードを記述することもできます。

次のようにして、ポートに対してリスナーを登録できます。

jp.go.aist.rtm.RTC.port.OutPortBase や jp.go.aist.rtm.RTC.port.InPortBase での jp.go.aist.rtm.RTC.port.ConnectorDataListenerT の取り扱いと同様です。

```
sdlEngine.local_ports["SDLEngine0.in"].getPortSupport().addConnectorDataListener(
jp.go.aist.rtm.RTC.port.ConnectorDataListenerType.ON_RECEIVED,
new jp.ac.kyutech.SRP.Scripting.ConnectorDataListener() {
    public void operator(jp.go.aist.rtm.RTC.port.ConnectorBase.ConnectorInfo info, RTC.TimedLong data){
        print("on received1");
    }
});
```

その後、SDLConsoleIn の out データポートに"1"を送信すると、

```
sdlConsoleIn.local_ports["SDLConsoleIn0.out"].put(1);
```

コンソールに次のように表示されます。

```
on received1
```

2.10. SDLEngine のデータポートのデモ2

先のデモはローカルのコンポーネントを利用しました。C++や Java のサンプルとして提供されている SimpleIO の ConsoleIn/ConsoleOut と繋いで動作することもできます。ここでは、C++の ConsoleOut と接続します。

1. 既に C++ 版の SimpleIO サンプルの ConsoleOutComp がビルド済みであると仮定します。ネームサーバと ConsoleOutComp を起動しておきます。以下の例では、rtc.conf に localhost:5005 でネームサーバを起動した場合のスクリプトです。また、naming も次のように設定してください。

```
corba.nameservers: localhost:5005
naming.formats: %n.rtc
```

2. build.xml の run-console ターゲットを起動して下さい。SDLEngine RT Component が作成されます。
3. 起動されたコンソールにて、次のスクリプトを動作して下さい。

```
env = rtc.env("localhost", 5005);
sdlEngine = rtc.local_component("SDLEngine", "SDLEngine");
java.lang.Thread.sleep(500);
env.get_handles();
env.connect(env.handles["SDLEngine0.rtc"].ports["SDLEngine0.out"],
            env.handles["ConsoleOut0.rtc"].ports["ConsoleOut0.in"]);
env.handles["SDLEngine0.rtc"].activate();
env.handles["ConsoleOut0.rtc"].activate();
```

4. 続けて次のスクリプトを実行します。

```
sdlEngine.local_ports["SDLEngine0.out"].put(1);
```

すると、SDLEngine の out データポートに"1"を送信し、結果として、ConsoleOut の in データポートで受信し、コンソールに次のように表示します。

```
Received: 1
```

2.11. 新しいデータポートを SDLEngine に組み込む

SDLEngine にデータポートを組み込む手順を紹介します。

1. build.xml の gen-rtc-sdl-engine ターゲットを修正します。

```
<target name="gen-rtc-sdl-engine">
    ....
    <arg value="--inport=in:RTC::TimedLong" />
    <arg value="--outport=out:RTC::TimedLong" />
    ....
</target>
```

2. ここでは、in という名の入力ポートと、out という名の出力ポートを作成しています。
3. データポートの定義は、サービスポートの定義と重複でき、入力も出力も同時に複数作成できます。
4. Ant の clean ターゲットと rtc.compile ターゲットを起動します。

以上の手順で、SDLEngine に データポートを組み込むことができます。

2.12. 独自のデータ型のデータポートを SDLEngine に組み込む

独自のデータ型を持つ IDL ファイルを作成し、それを SDLEngine にデータポートとして組み込む手順を紹介します。

1. IDL ファイルを SDLEngine/main/idl に配置します。ここでは、MyData.idl とします。
2. build.xml の gen-rtc-sdl-engine ターゲットを修正します。

```
<arg value="--inport=myDataIn:jp::ac::kyutech::SRP::demo::MyData"/>
<arg value="--outport=myDataOut:jp::ac::kyutech::SRP::demo::MyData" />
```

3. Ant の clean ターゲットと rtc.compile ターゲットを起動します。

以上の手順で、SDLEngine に IDL ファイルで定義されたデータポートを組み込むことできます。

4. 次のスクリプトを独自データ型の送受信を確認するために用意します。

```
env = rtc.env("localhost", 5005);
sdlEngine = rtc.local_component("SDLEngine", "SDLEngine");
env.get_handles();
env.connect(env.handles["SDLEngine0.rtc"].ports["SDLEngine0.myDataIn"],
env.handles["SDLEngine0.rtc"].ports["SDLEngine0.myDataOut"]);
env.handles["SDLEngine0.rtc"].activate();
sdlEngine.local_ports["SDLEngine0.myDataIn"].addListener(new
jp.ac.kyutech.SRP.Scripting.InPortListener() {
    dataReceived(event) {
        print("Received: " + event.getValue().data1 + "," + event.getValue().data2);
    }
});
```

5. 次のスクリプトを実行すると、SDLEngine の myDataOut データポートにデータを送信する。

```
sdlEngine.local_ports["SDLEngine0.myDataOut"].put(new  jp.ac.kyutech.SRP.demo.MyData(1, "abc"));
```

結果として、SDLEngine の myDataIn データポートで受信し、コンソールに結果を表示します。

```
Received: 1,abc
```

2.13. 新しい RT コンポーネントを SDLEngine（データポート）に対応させる

ここでは、データポートに対応するための手順を示します。

1. build.xml の gen-rtc-sdl-console-in ターゲットを元に、Ant のターゲットを追加します。
2. build.xml の rtc.all ターゲットに、作成した gen-rtc-* ターゲットを追加します。

```
<target name="rtc.all" description="Generate RTComponents code.">
    ....
    <!-- Append rtc generation target -->
    <antcall target="gen-rtc-sdl-engine" />
    <antcall target="gen-rtc-foo" />
    <!-- Append rtc generation target -->
    ....
</target>
```

3. Ant の clean ターゲットと rtc.compile ターゲットを起動します。

```
$ ant clean rtc.compile
```

以上の手順で、データポート RT Component が作成できます。

2.14. OpenHRI と SDLEngine を接続したデモ

SDLEngine から OpenHRI に接続して、音声発話や音声認識を行う手順を示します。OpenRTM の基本的な導入は完了しているものとします。

(1) OpenHRI のインストールと初期準備

1. Windows に OpenHRI をインストールします。update-checker-1.01.exe を実行すると必要なパッケージのインストールまたはアップデートをまとめて行えます。

2. OpenHRI の設定ファイルを編集して、CORBA ネームサーバや名前形式を調整して、SDLEngine と一致させておきます

C:¥Program Files (x86)¥OpenHRIAudio¥rtc.conf の以下の項目を編集します。

```
corba.nameservers: localhost:5005
naming.formats: %n.rtc
```

C:¥Program Files (x86)¥OpenHRIVoice¥rtc.conf の以下の項目を編集します。

```
corba.nameservers: localhost:5005
naming.formats: %n.rtc
```

※ OpenHRIVoice¥rtc.conf に corba.endpoint: の記述があると起動しないようです。

また、ネームサーバを起動しておきます。

(2) 発話をするまでの手順

1. SDLEngine に RTC.TimedString 型の出力データポートを用意します。配布時点では、既にこのデータポートは用意されていますので通常は不要です。

build.xml に 次のコード断片を追加し、stringOut という名のデータポートを作成します。

```
<arg value="--outport=stringOut:RTC::TimedString" />
```

2. SDLEngine/build.xml を右クリックして、実行→Ant ビルドを実行して下さい。
3. 問題なくビルドが完了しますと、"SDLEngine Console"というウィンドウが表示されます。
4. スタートメニューから OpenHRI→audio→portaudiooutput を選択して起動します。
5. スタートメニューから OpenHRI→voice→openjtalkrtc を起動します。
6. 以下のスクリプトを SDLEngine Console で実行します。

```
env = rtc.env("localhost", 5005);
sdlEngine = rtc.local_component("SDLEngine", "SDLEngine");
env.get_handles();
env.connect(env.handles["SDLEngine0.rtc"].ports["SDLEngine0.stringOut"],
env.handles["OpenJTalkRTC0.rtc"].ports["OpenJTalkRTC0.text"]);
env.connect(env.handles["OpenJTalkRTC0.rtc"].ports["OpenJTalkRTC0.result"],
   env.handles["PortAudioOutput0.rtc"].ports["PortAudioOutput0.AudioDataIn"]);
env.handles["SDLEngine0.rtc"].activate();
env.handles["OpenJTalkRTC0.rtc"].activate();
env.handles["PortAudioOutput0.rtc"].activate();
sdlEngine.local_ports["SDLEngine0.stringOut"].put("こんにちわ");
```

7. "こんにちは" と発話されれば OK です。

OpenHRIAudioManager から portaudiooutput を起動した場合以下のようなエラーが出ます
// Error: Fail to create the handle: XXXXXXXX.host_cxt
これを避けるには、portaudiooutput を直接起動してください。

(3) 音声認識をする

1. SDLEngine に RTC.TimedString 型の出力データポートを用意します。配布時点では、既にこのデータポートは用意されていますので通常は不要です。

build.xml に 次のコード断片を追加し、stringIn という名のデータポートを作成します。

ここで音声認識の結果を受け取ります。

```
<arg value="--inport=stringIn:RTC::TimedString" />
```

2. SDLEngine/build.xml を右クリックして、実行 => Ant ビルドを実行して下さい。
3. 問題なくビルドが完了しますと、"SDLEngine Console"というウィンドウが表示されます。
4. スタートメニューから OpenHRI→audio→portaudioinput を選択して起動します。
5. スタートメニューから OpenHRI→voice→juliusrtc を起動します。

グラマーファイルを選択するダイアログが開きますので、ワークスペース内に用意されている SDLEngine/doc/robot.grxml を選択します。

6. 以下のスクリプトを SDLEngine Console で実行します。

```
env = rtc.env("localhost", 5005);
sdlEngine = rtc.local_component("SDLEngine", "SDLEngine");
env.get_handles();
env.connect(env.handles["SDLEngine0.rtc"].ports["SDLEngine0.stringIn"],
env.handles["JuliusRTC0.rtc"].ports["JuliusRTC0.result"]);
env.connect(env.handles["JuliusRTC0.rtc"].ports["JuliusRTC0.data"],
env.handles["PortAudioInput0.rtc"].ports["PortAudioInput0.AudioDataOut"]);
env.handles["SDLEngine0.rtc"].activate();
env.handles["JuliusRTC0.rtc"].activate();
env.handles["PortAudioInput0.rtc"].activate();
sdlEngine.local_ports["SDLEngine0.stringIn"].addListener(new jp.ac.kyutech.SRP.Scripting.InPortListener() {
  dataReceived(event) {
    print("Received: " + event.getValue().data);
  }
});
```

7. "クッキーを取って" とマイクに向かって話して、結果が XML として受け取れれば OK です。

OpenHRIAudioManager から portaudioinput を起動した場合以下のようなエラーが出ます

// Error: Fail to create the handle: XXXXXXXX.host_cxt

これを避けるには、portaudioinput を直接起動してください。

2.15. GUI を伴わない SDLEngine の起動

GUIのコンソールなしでSDLEngineを起動して、スクリプトを実行する手順です。

1. SDLEngineをコンパイルしておきます。

```
ant clean rtc.compile
```

2. build.xml を調整して、読み込むスクリプトを調整します。

以下での〈arg〉タグを調整します。

```
<target name="run-main" depends="init,rtc.compile" description="Run the BeanShell Console">
    <java classname="jp.ac.kyutech.SRP.Main"  fork="true">
        <classpath refid="libs.path" />
        <classpath location="${main.build.classes.dir}" />
        <classpath location="${rtc.build.classes.dir}" />
        <arg value="doc/test02.txt"/>
    </java>
</target>
```

この例では、SDLEngine/doc/test02.txt を実行します。

3. 必要なコンポーネントの起動などを済ませます。

このテストスクリプトをそのまま使用する場合は、 OpenHRI の portaudiooutput と OpenJTalk 起動しておきます。

4. SDLEngineをGUI無しで起動します。

```
ant run-main
```

5. 「test」を発音しているのが聞こえたら成功です。

なお、終了時に警告がでますが、問題ありません。恐らく、OpenRTM-aist-java の実装によるものと思われます。