ROBOTECH
RTミドルウェア講習会

日時：2012年7月11日(水)　～ 13日(金)
場所：東京ビッグサイト東2ホール ROBOTECH内 ROBOTECHフィールド

RT MIDDLEWARE

RTミドルウエア 人型ロボットワークショップ

(独)産業技術総合研究所
原　　功

この講習での主な内容

- 講義
  - ChorenoidとOpenHRIに関する概要説明
- 実習
  - G-ROBOT　GR001を使った音声コマンドシステムの構築
  - G-ROBOT　GR001のオリジナルの動作パターンの作成
  - KINECTをつかってG-ROBOT GR001を操作する

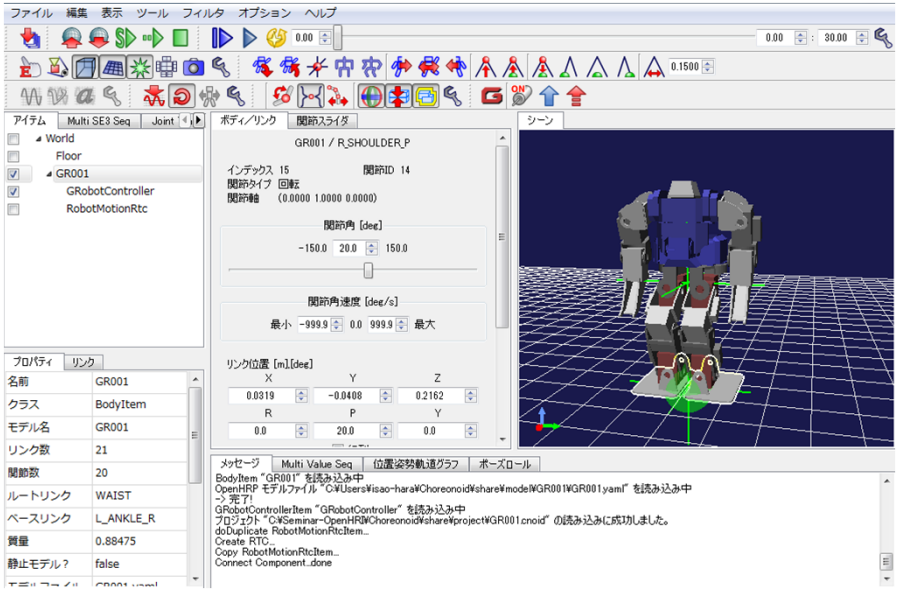
この講習での主な内容

- 講義
  - ChorenoidとOpenHRIに関する概要説明
- 実習
  - G-ROBOT　GR001を使った音声コマンドシステムの構築
  - G-ROBOT　GR001のオリジナルの動作パターンの作成
  - KINECTをつかってG-ROBOT GR001を操作する

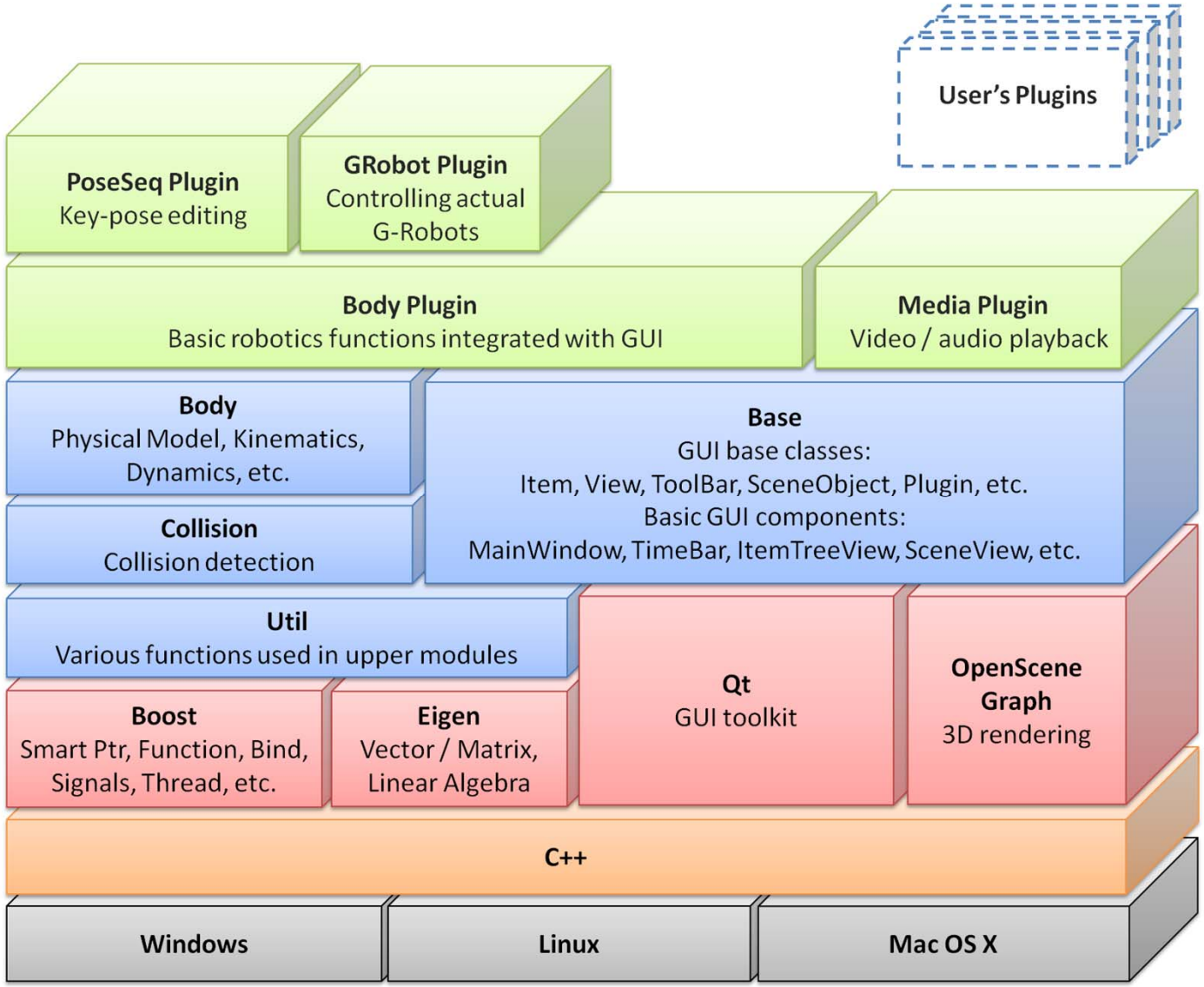
Choreonoid

多関節型ロボットの動作パターンを作成するためのGUIツール

- キーフレームベースの姿勢設定と動作補完
  - ユーザは、キーポーズを作成するだけ
- 姿勢設定時に動力学シミュレーションを同時実行
  - 無理な姿勢を自動的に修正
- C++による高速な処理の実現
  - より高速に、より安定に
- プラグインにより様々な機能拡張が可能
  - より柔軟に、拡張可能に
    - 動力学シミュレーション
    - ロボット操作RTC



Choreonoidのシステム構成

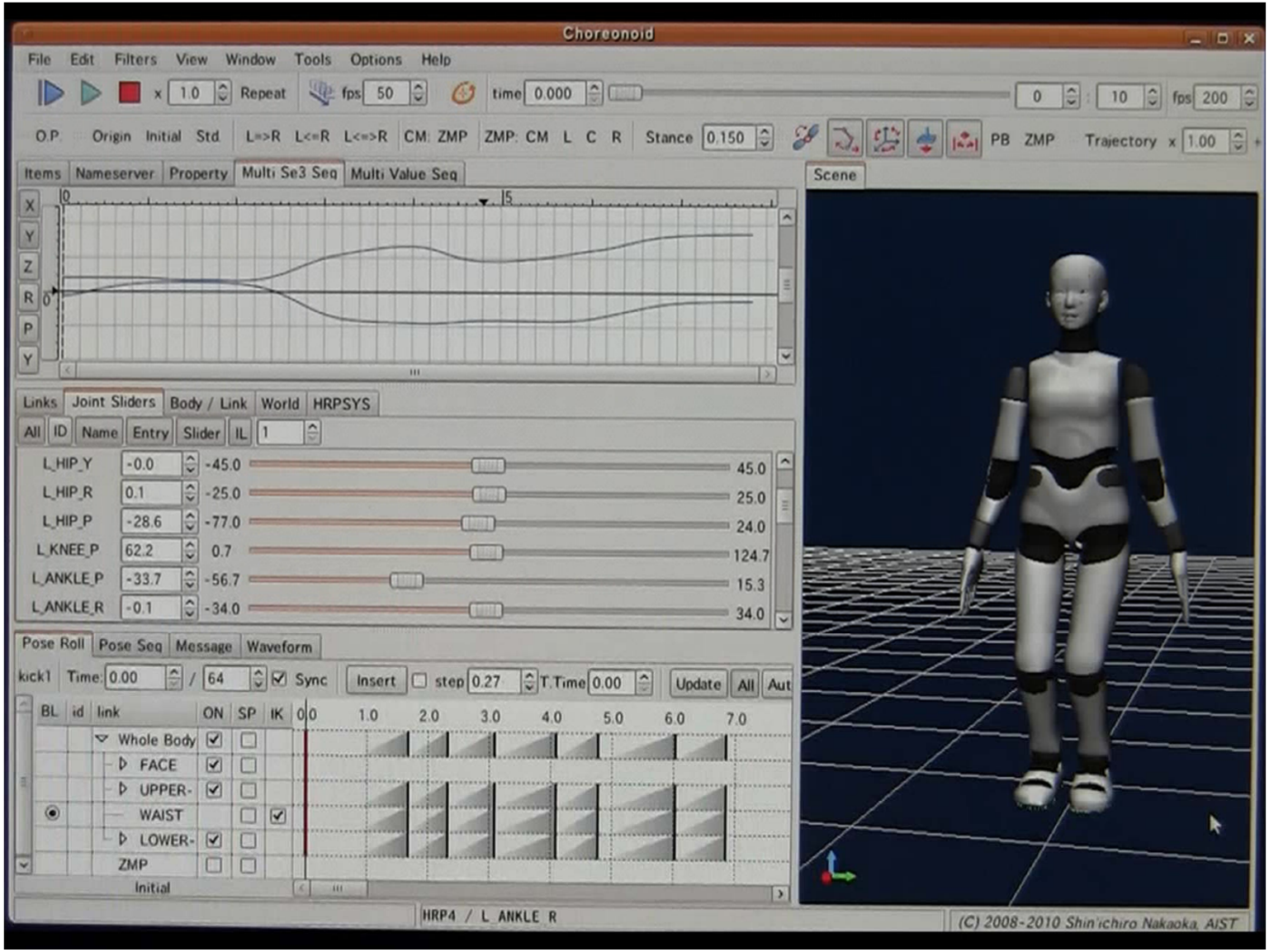


←プラグイン

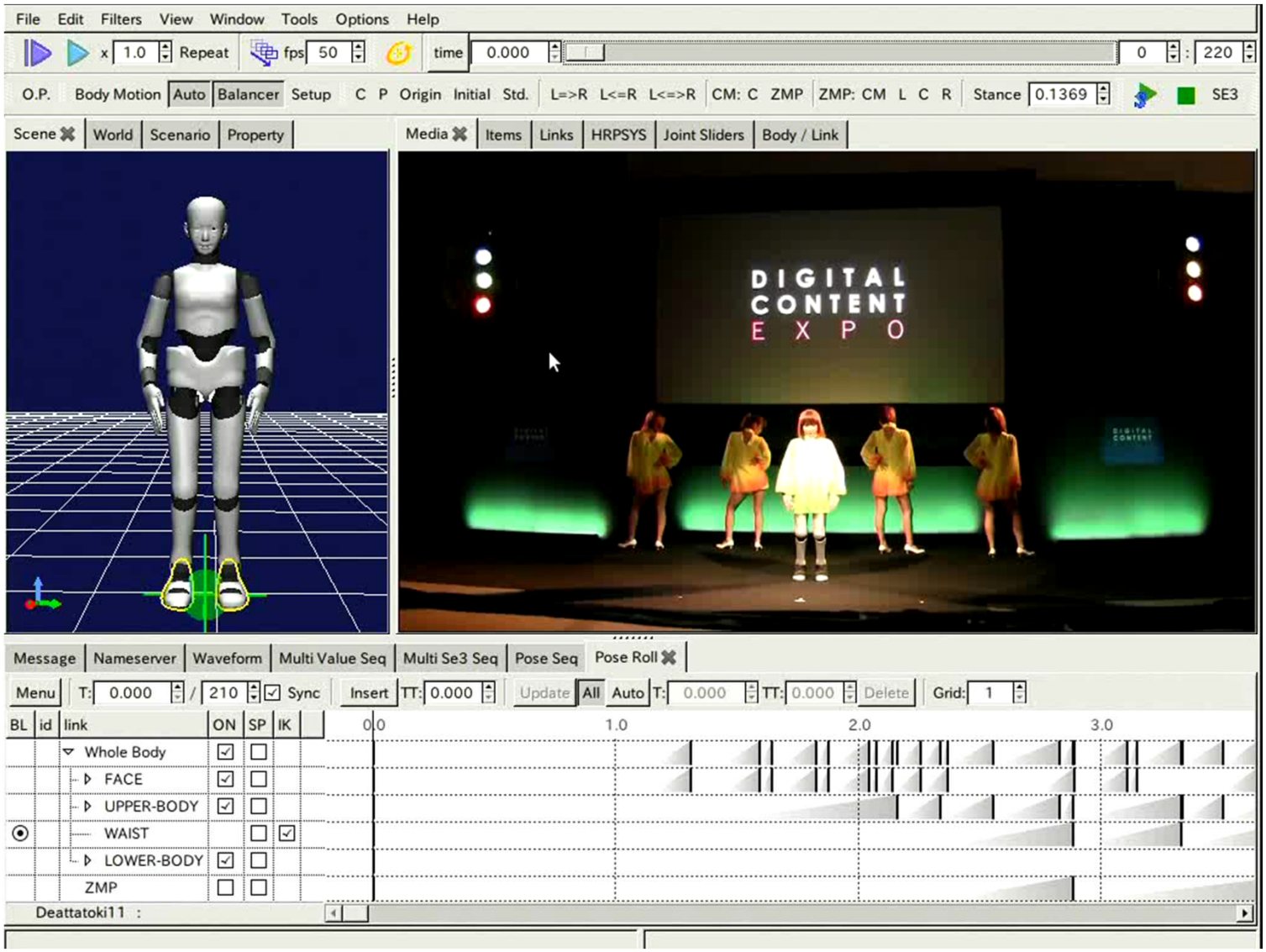
←基本モジュール

←外部ライブラリ

Choreomoidの活用事例（HRP-4C）

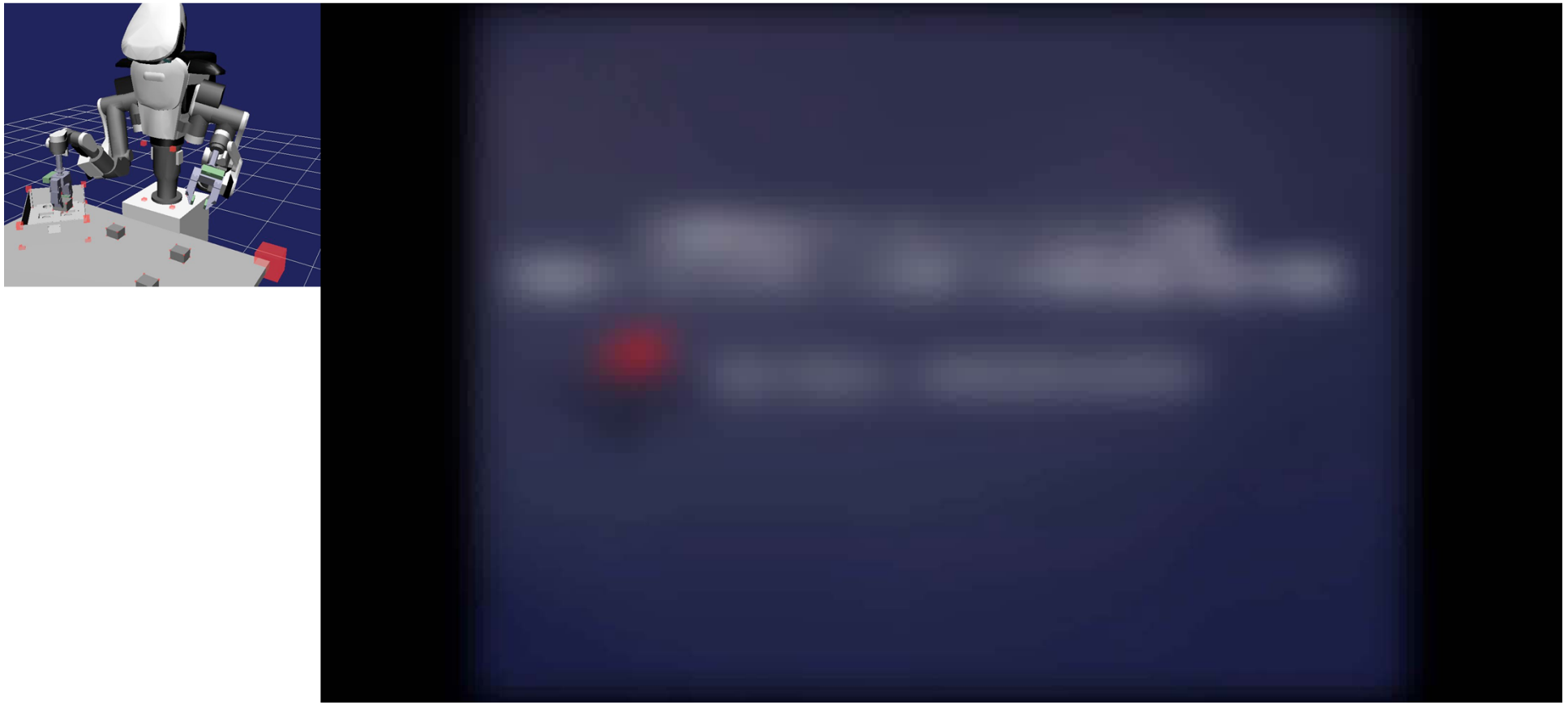


Choreonoidの活用事例(HRP-4C)



Choreonoidの活用事例

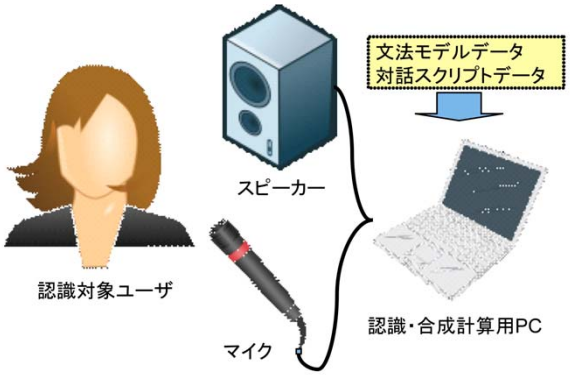
- 頭部ステレオカメラを用いた双腕ロボットによるマニピュレーション作業
  - graspPlugin for Choreonoid
  - **http://choreonoid.org/GraspPlugin/**



コミュニケーション知能モジュール群 OpenHRI

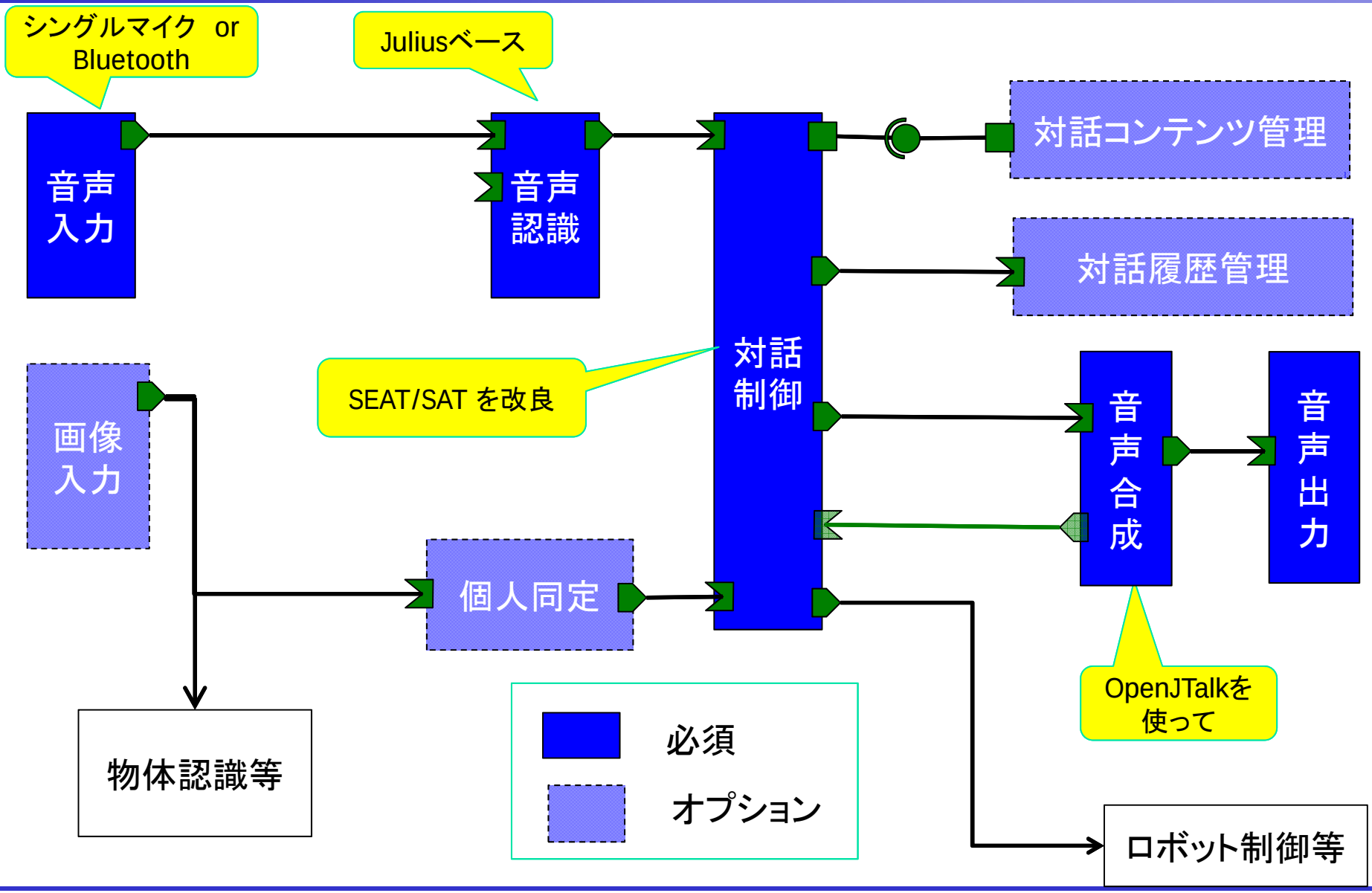
- 音声認識・音声合成・対話制御など、ロボットのコミュニケーション機能の実現に必要な各要素を実現するコンポーネント群
- 知能化PJ　コミュニケーション知能共通規格準拠

  - **OpenHRIAudio　(14モジュール)**
    - マイク入力、スピーカー出力、エコー除去などの音響処理
  - **OpenHRIVoice (3モジュール)**
    - 音声認識、音声合成
  - **SEATSAT　(2モジュール)**
    - 対話制御
  - **その他（1モジュール）**
    - **KINECTモジュール**



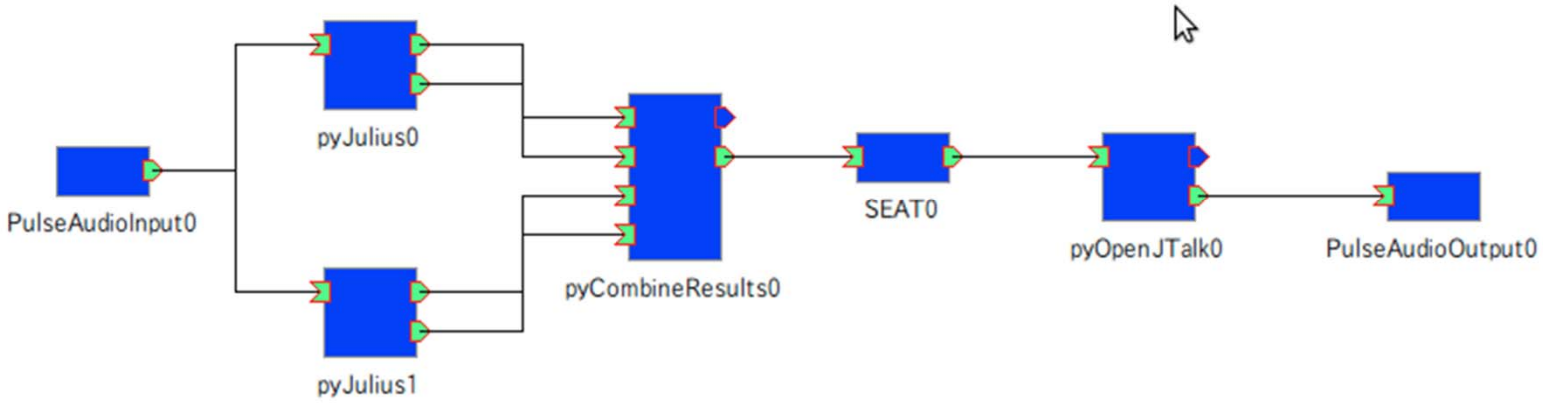
- 現在は、OpenRTC-aistの一部としてメンテナンスを継続

OpenHRIで提供する機能

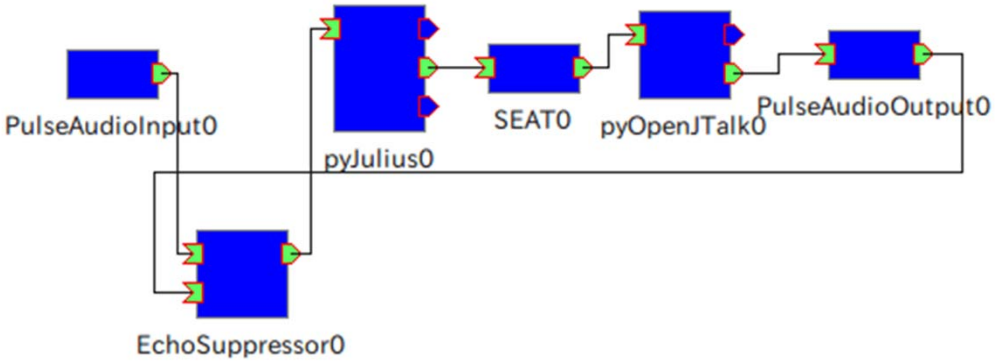


OpenHRIを使ったシステム例

- マルチリンガル対話システム



- 音響エコー除去機能を有する対話システム構成



この講習での主な内容

- 講義
  - ChorenoidとOpenHRIに関する概要説明
- 実習
  - G-ROBOT　GR001を使った音声コマンドシステムの構築
  - G-ROBOT　GR001のオリジナルの動作パターンの作成
  - KINECTを使ってG-ROBOT GR001を操作する

この講習で使うソフトウェア

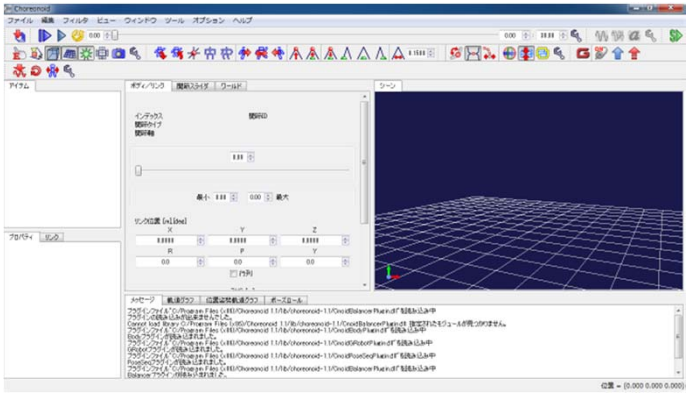
準備

- KINECTを使う場合には、KINECT　SDK 1.5 をインストール
  - KINECT　SDKは、Windows7が必要なので確認
  - USBメモリのKinectSDK-v1.5-Setup.exeを実行し、指示に従いインストールする
- OpenRTM-aist-1.1.0-Releaseをインストール
  - USBメモリ内のdownloadのフォルダにOpenRTM-aist-1.1.0-RELEASE_vc10.msiで導入する。
- USBシリアルケーブルを接続して、ドライバをインストール
  - USBメモリの　¥USB-Serial Driver 内にある。
  - デバイスマネージャーでシリアルポートがあることを確認。
- Choreonoid、OpenHRIなどをインストール
  - 通常は、各ソフトウェアのインストーラで行うが、今回はUSBメモリから導入する
  - USBメモリのSeminar-OpenHRIを C:¥にコピー
  - Seminar-OpenHRI には、Choreonoid1.1, OpenHRI等の必要なモジュールを収録済み

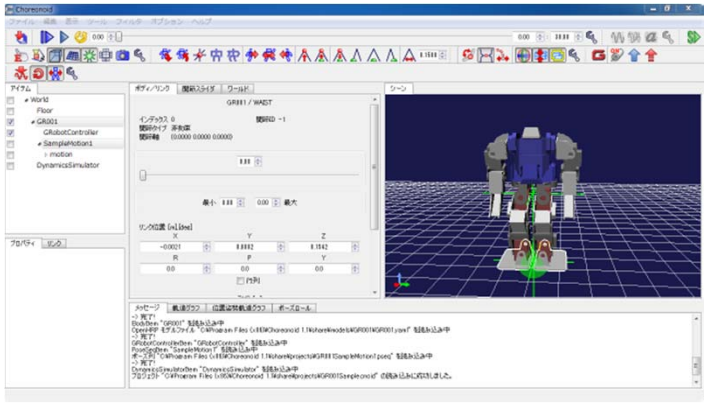
この講習で使うソフトウェアの確認

Chorenoidを起動してみる

- C:¥Seminar-OpenHRIの下にある choreonoid-1.1のショートカットから Choreonoidの起動を確認する。
- 次に、G-ROBOTのサンプルプロジェクトを読み込む
  - C:¥Seminar-OpenHRI¥Choreonoid-1.1¥share¥projects¥GR001Sample.cnoid
- サンプル動作の実行
- G-ROBOT GR001を接続して動作することを確認する



起動直後

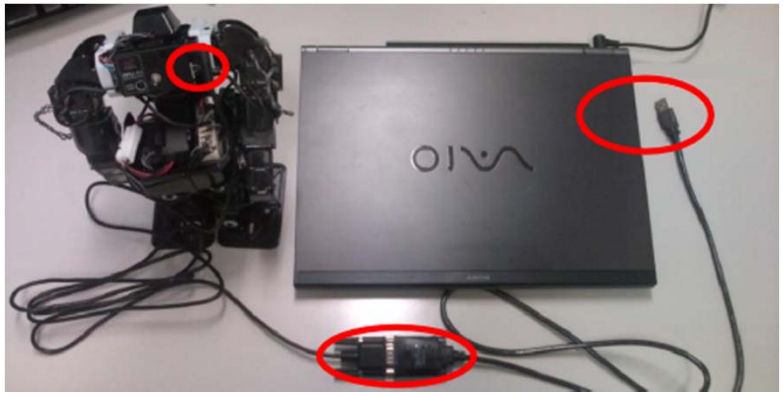


サンプルプロジェクト読み込み後

この講習で使うソフトウェアの確認

- G-ROBOT GR001を接続して動作することを確認する

①接続



②ポートの設定

←ここを修正

③接続確認

↑ここのボタンを押してサーボモータのON/OFFを確認

この講習で使うソフトウェアの確認

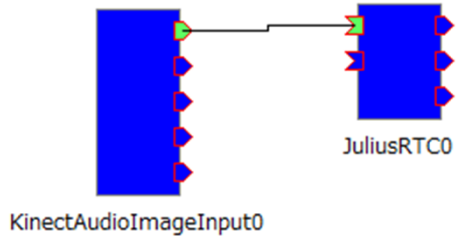
ChorenoidのRobotMotionRtcの動作を確認

- スタート → OpenRTM-aist 1.1 → C++ → tools → Start Naming Service　で ネームサーバーを起動する。
- C:¥Seminar-OpenHRIの下にある choreonoid-1.1のショートカットから Choreonoidの起動する。
- G-ROBOTのプロジェクトを読み込む
  - C:¥Seminar-OpenHRI¥Choreonoid-1.1¥share¥projects¥GR001.cnoid
- RTシステムエディタを起動し、NameServiceビューに　Cnoid_RobotMotion0|rtc があることを確認

この講習で使うソフトウェアの確認

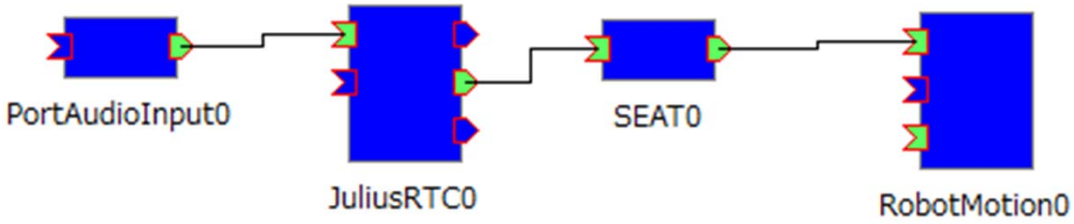
- KINECTコンポーネントの動作確認
  - スタート → OpenRTM-aist 1.1 → C++ → tools → Start Naming Service でネームサーバーを起動する。
  - C:¥Seminar-OpenHRIのフォルダ内のKinectRTCというショートカットを起動する。起動完了まで4秒待つ必要がある。
  - スタート → OpenRTM-aist 1.1 → C++ → tools → RTSystemEditor でRTシステムエディタを起動しKINECTコンポーネントが起動していることを確認する。

- Juliusrtcを起動して、KINECTコンポーネントを接続し、音声認識できていることを確認する。
  - C:¥Seminar-OpenHRIのフォルダ内のjuliusrtcというショートカットでコンポーネントを起動する。文法ファイルの選択ダイアログがでてきたら、C:¥Seminar-OpenHRI¥Sample内のsimple_demo.grxmlを選択。
  - RTシステムエディタで起動を確認し、下図のように接続し、アクティベートして「ばいばい」、「さようなら」、「こんにちは」、「きねくと」が認識できることをコンソールで確認。



G-ROBOTを使った音声コマンドシステムの構築

- KINECTを音声入力デバイスとして利用する。
- 下記のコンポーネントを起動して、RTシステムエディタで結線する
  - KinectAudioImageInputRTC
  - JuliusRTC
    - simple_demo.grxmlを読み込む
  - SEAT
    - simple_demo.seatmlを読み込む
  - Choreonoid
    - プロジェクトファイルGR001.cnoidを読み込む



- 「ばいばい」、「こんにちは」を発話し、動作を確認する。

音声コマンドを拡張しよう

音声コマンドを拡張しよう

- JulisuRTCで使うgrxmlファイル（音声認識文法ファイル）を編集して認識できる言葉を追加

- SEATで使うseatmlファイル（対話ルール記述ファイル）にルールを追加

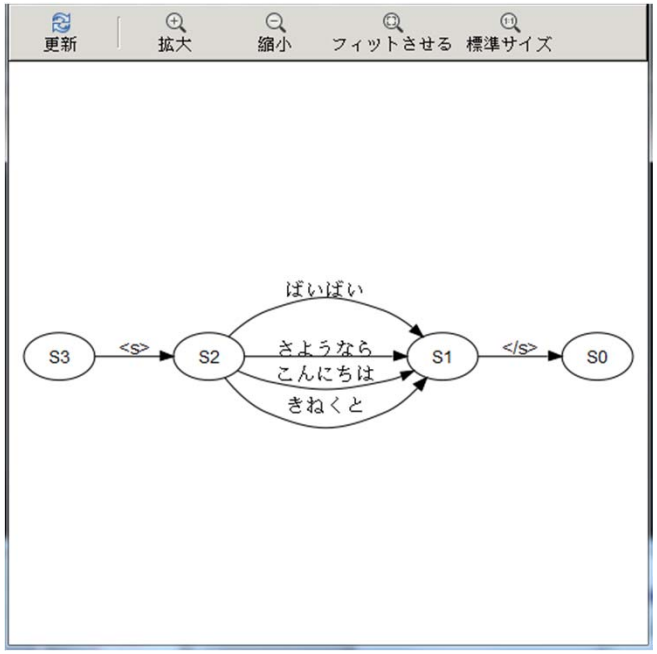
音声認識文法ファイル

JulisuRTCで使うgrxmlファイル（ W3C-SRGS形式）

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                        http://www.w3.org/TR/speech-grammar/grammar.xsd"
        xml:lang="jp"
        version="1.0" mode="voice" root="command">

  <rule id="command">
    <one-of>
      <item><ruleref uri="#command_simple" /></item>
    </one-of>
  </rule>

  <rule id="command_simple">
    <one-of>
      <item>ばいばい</item>
      <item>さようなら</item>
      <item>こんにちは</item>
      <item>きねくと</item>
    </one-of>
  </rule>
</grammar>
```



Srgeditorで文法を確認

音声認識文法ファイル

- OpenHRIの音声認識RTCでは、W3C-SRGS形式の音声認識文法を使用
- W3C-SRGSのタグ
  - lexicon: W3C-PLS辞書(次のセクション)のURIを定義します。任意。
  - rule: IDによって区別された各文法を定義します。IDは音声認識文法の相互参照や、Julius音声認識コンポーネントによって認識されるアクティブな文法を 切り換えるのに利用します。
  - item: 認識される単語や文を定義します。repeatプロパティで繰り替えされる回数を指定できます。
  - one-of: 子項目で定義される文法がすべて許容できることを示します。
  - ruleref: uriで指定される文法を参照します

音声コマンドを拡張しよう

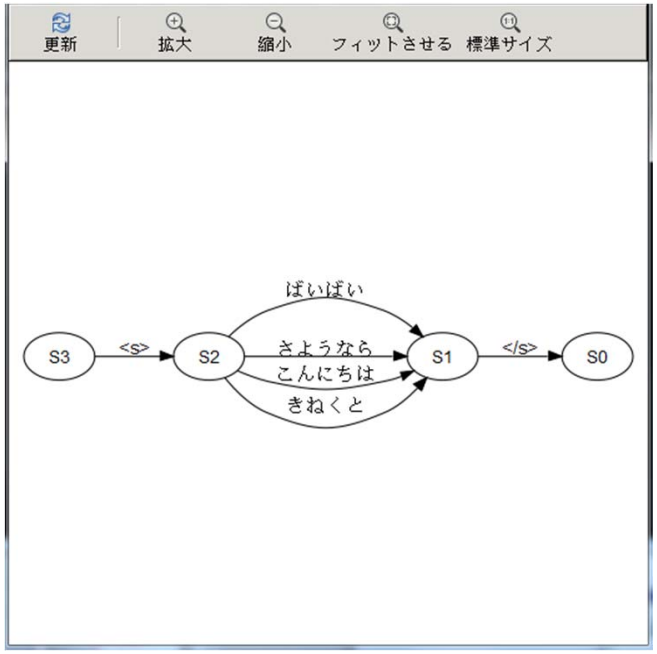
- 音声コマンドを拡張しよう
  - JulisuRTCで使うgrxmlファイルを編集して認識できる言葉を追加
  - SEATで使うseatmlファイルにルールを追加

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar xmlns="http://www.w3.org/2001/06/grammar"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2001/06/grammar
                        http://www.w3.org/TR/speech-
grammar/grammar.xsd"
        xml:lang="jp"
        version="1.0" mode="voice" root="command">

  <rule id="command">
    <one-of>
      <item><ruleref uri="#command_simple" /></item>
    </one-of>
  </rule>

  <rule id="command_simple">
    <one-of>
      <item>ばいばい</item>
      <item>さようなら</item>
      <item>こんにちは</item>
      <item>きねくと</item>
    </one-of>
  </rule>
</grammar>
```

ここに認識する言葉を追加



srgeditorで確認

音声コマンドを拡張しよう

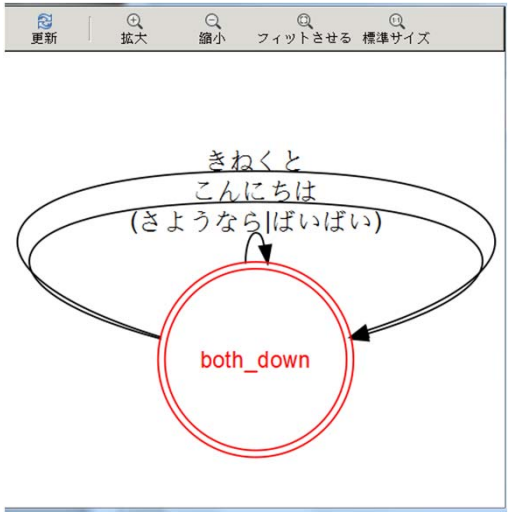
SEATで使うseatmlファイル（対話ルール）

```
<?xml version="1.0" encoding="UTF-8"?>
<seatml>
  <general name="flaggame">
    <agent name="speechin" type="rtcin" datatype="TimedString" />
    <agent name="command" type="rtcout" datatype="TimedString" />
  </general>

  <state name="both_down">
    <rule>
      <key>(さようなら|ばいばい)</key>
      <command host="command">bye</command>
    </rule>
    <rule>
      <key>こんにちは</key>
      <command host="command">bow</command>
    </rule>
    <rule>
      <key>きねくと</key>
      <command host="command">KINECT</command>
    </rule>
  </state>

</seatml>
```

データポートの定義



seateditorでルールを確認

対話制御スクリプト　SEATML

SEATMLは、シンプルな対話エンジンSEATの動作を定義するためのXMLファイル。状態ごとの＜条件ーアクション＞のルールを記述し、条件に適合した場合の動作を記述したもの。

- アダプタの定義
  - SEATには、名前と通信方法(RTMとソケット)を対応付けるアダプタ機構を持っています。アダプタ機構は、通信方法の差異を隠蔽化することで、システムのハードウェア構成の変化に適応し、対話ロジックの再利用性を向上させます。
  - General：アダプタ定義部を示します。
  - Agent：名前と通信方法の対応を示します。“type”属性は”rtcin”、”rtcout”、”socket”を取ることができます。タイプが”rtcin”か”rtcout”と定義されたとき、”datatype”属性を定義できます(データ型に関しては、RTMの仕様を参照してください)。タイプが”socket”と定義されたとき、”host” 、”port” 属性を定義できます。
- スクリプト定義
  - State: 状態遷移モデルで状態を示します。
  - Rule：キーワードとコマンドの組を定義します。
  - Key：キーワードを示します。
  - Command：キーワードと入力が一致したとき実行されるコマンドを示します。
  - Statetransition：状態遷移を示します。

音声コマンドを拡張しよう

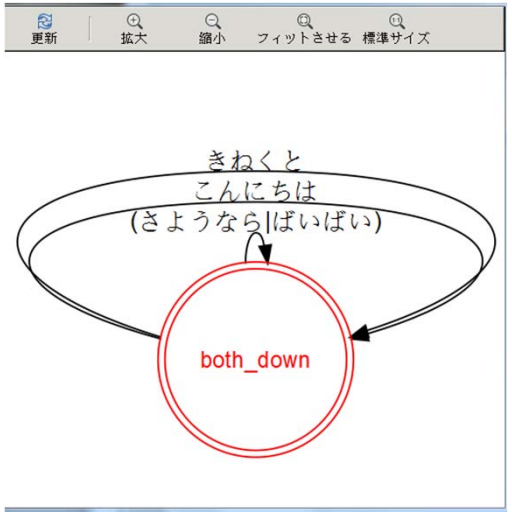
- 音声コマンドを拡張しよう
  - JulisuRTCで使うgrxmlファイルを編集して認識できる言葉を追加
  - SEATで使うseatmlファイルにルールを追加

```
<?xml version="1.0" encoding="UTF-8"?>
<seatml>
  <general name="flaggame">
    <agent name="speechin" type="rtcin" datatype="TimedString" />
    <agent name="command" type="rtcout" datatype="TimedString" />
  </general>

  <state name="both_down">
    <rule>
      <key>(さようなら|ばいばい)</key>
      <command host="command">bye</command>
    </rule>
    <rule>
      <key>こんにちは</key>
      <command host="command">bow</command>
    </rule>
    <rule>
      <key>きねくと</key>
      <command host="command">KINECT</command>
    </rule>
  </state>

</seatml>
```

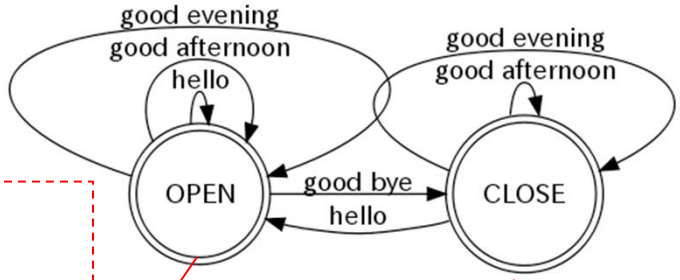
ここにルールを追加



seateditorで確認

SEATMLの例（状態遷移あり）

```
<?xml version="1.0" encoding="UTF-8"?>
 <seatml>
  <general name="sample">
   <agent name="speechin" type="rtcin" datatype="TimedString" />
   <agent name="speechout" type="rtcout" datatype="TimedString" />
 </general>
 <state name="OPEN">
  <rule>
   <key>hello</key> <command host="speechout">Hello.</command>
  </rule>
  <rule>
   <key>good afternoon</key> <command host="speechout">Good afternoon.</command>
  </rule>
  <rule>
   <key>good evening</key> <command host="speechout">Good evening.</command>
  </rule>
  <rule>
   <key>good bye</key> <command host="speechout">Good bye.</command>
   <statetransition>CLOSE</statetransition>
  </rule>
 </state>
 <state name="CLOSE">
  <rule>
   <key>hello</key> <command host="speechout">Hello there.</command>
   <statetransition>OPEN</statetransition>
  </rule>
  <rule>
   <key>good afternoon</key> <command host="speechout">I'm not available.</command>
  </rule>
  <rule>
   <key>good evening</key> <command host="speechout">I'm not available.</command> </rule>
 </state>
</seatml>
```

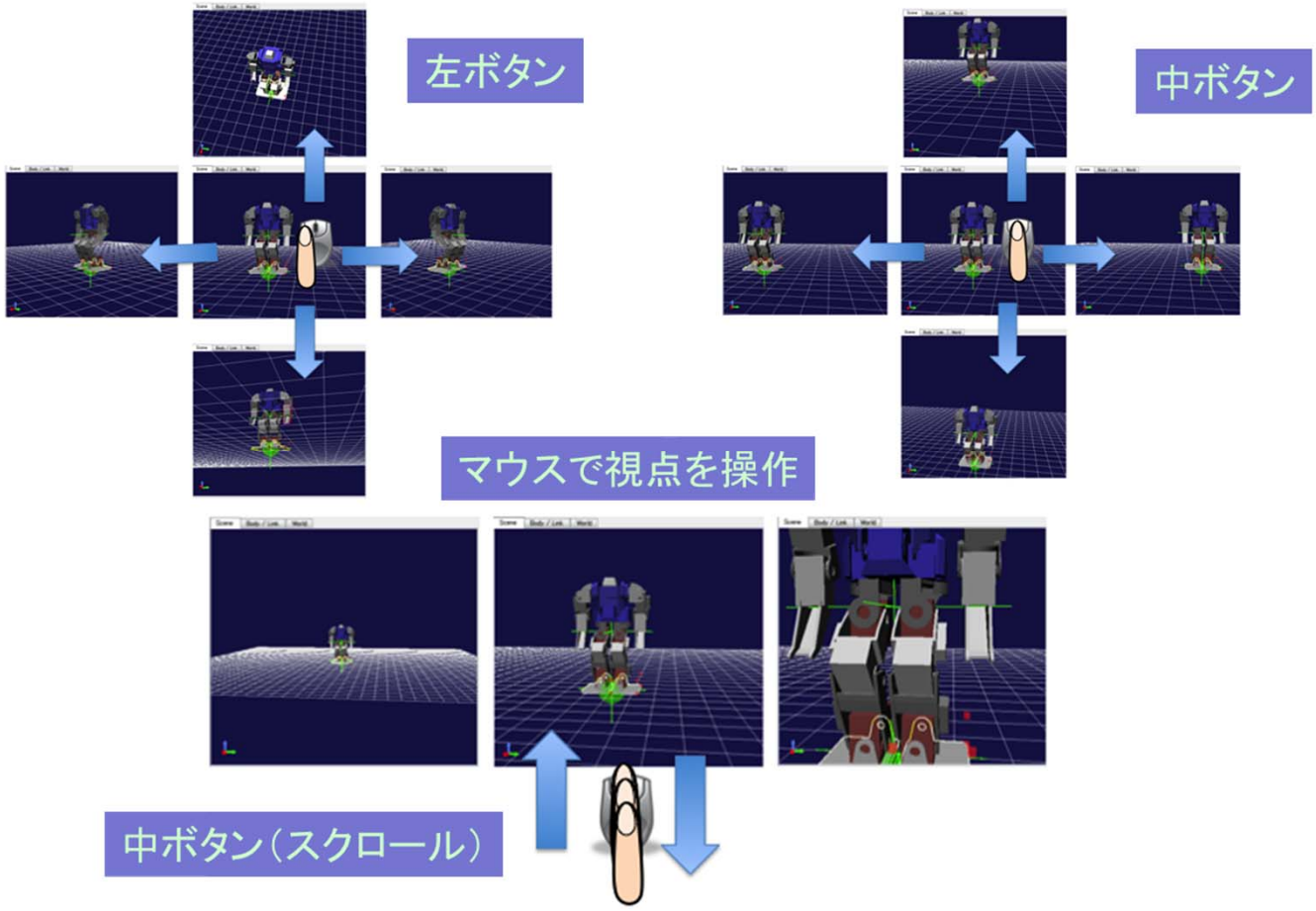


音声コマンドを拡張しよう

- Choreonoidで実行可能なモーション
  - Chreonoid/share/motions の下にある　.yaml ファイル
    - b_step ------ 一歩下がる
    - F_step ------ 一歩前に出る
    - Balance ------- 片足バランス
    - gymnastics ------- 体操
    - kick -------- キックの動作
    - muri -------- 腕を前で振る
    - bow --------- お辞儀する
    - bye -------- 手を振る
    - init -------- 直立姿勢
    - rest --------- 両膝をまげて歩行準備姿勢
    - SampleMotion1 ---------- サンプル動作
    - borthup[down] --------- 両腕を上げる[下げる]
    - leftup1[up2] --------- 左腕を上げる
    - leftdown1[down2] ------- 左腕を下げる
    - rightup1[up2] -------- 右腕を上げる
    - lrightdown1[down2] ----- 右腕を下げる

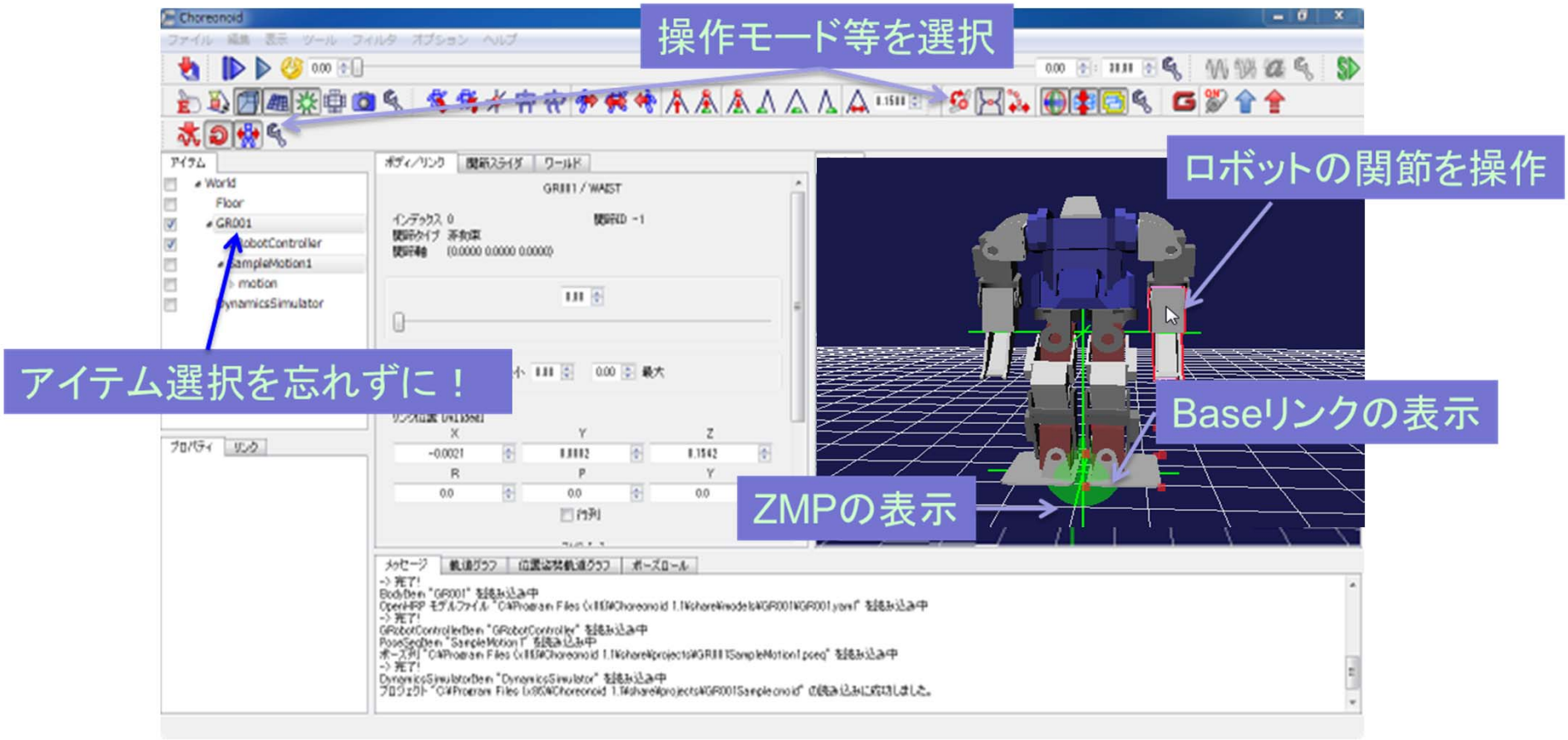
G-ROBOT　GR001のオリジナルの動作パターンの作成

- Choreonoidで新しい動作パターンを作る
  - Choreonoidの基本的な操作方法



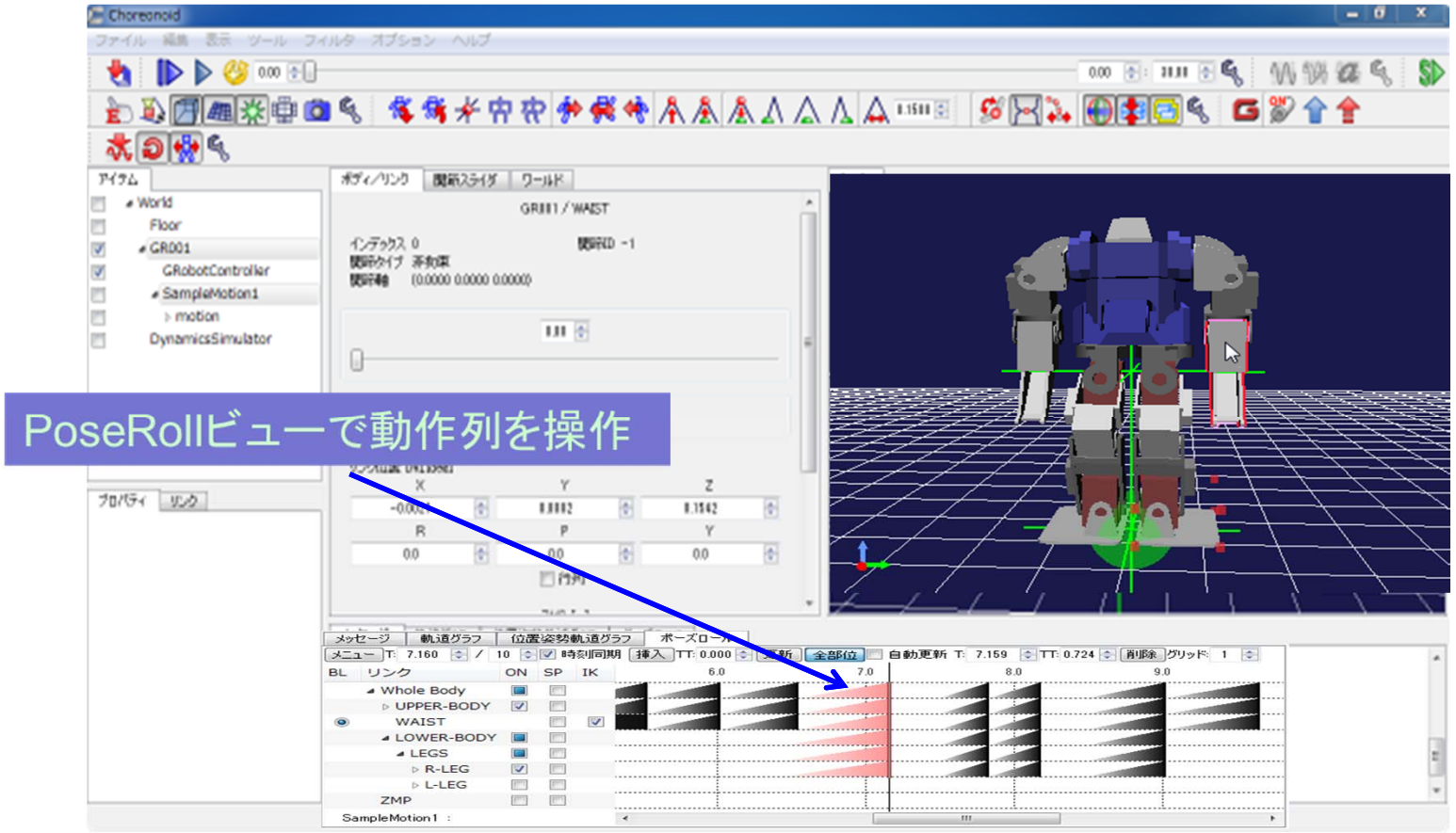
G-ROBOT　GR001のオリジナルの動作パターンの作成

- Choreonoidで新しい動作パターンを作る
  - Choreonoidの基本的な操作方法

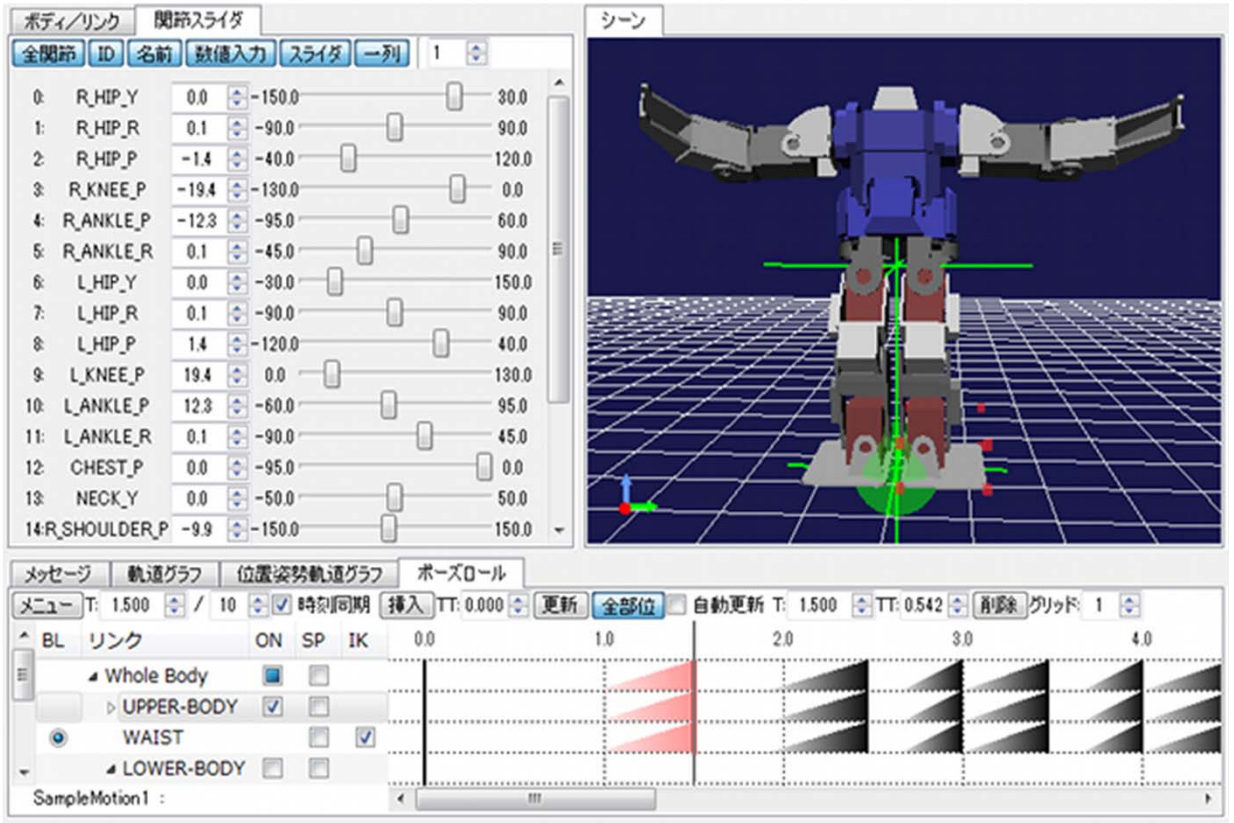


G-ROBOT　GR001のオリジナルの動作パターンの作成

- Choreonoidで新しい動作パターンを作る
  - 既存の動作パターンを読み込んで、編集する。



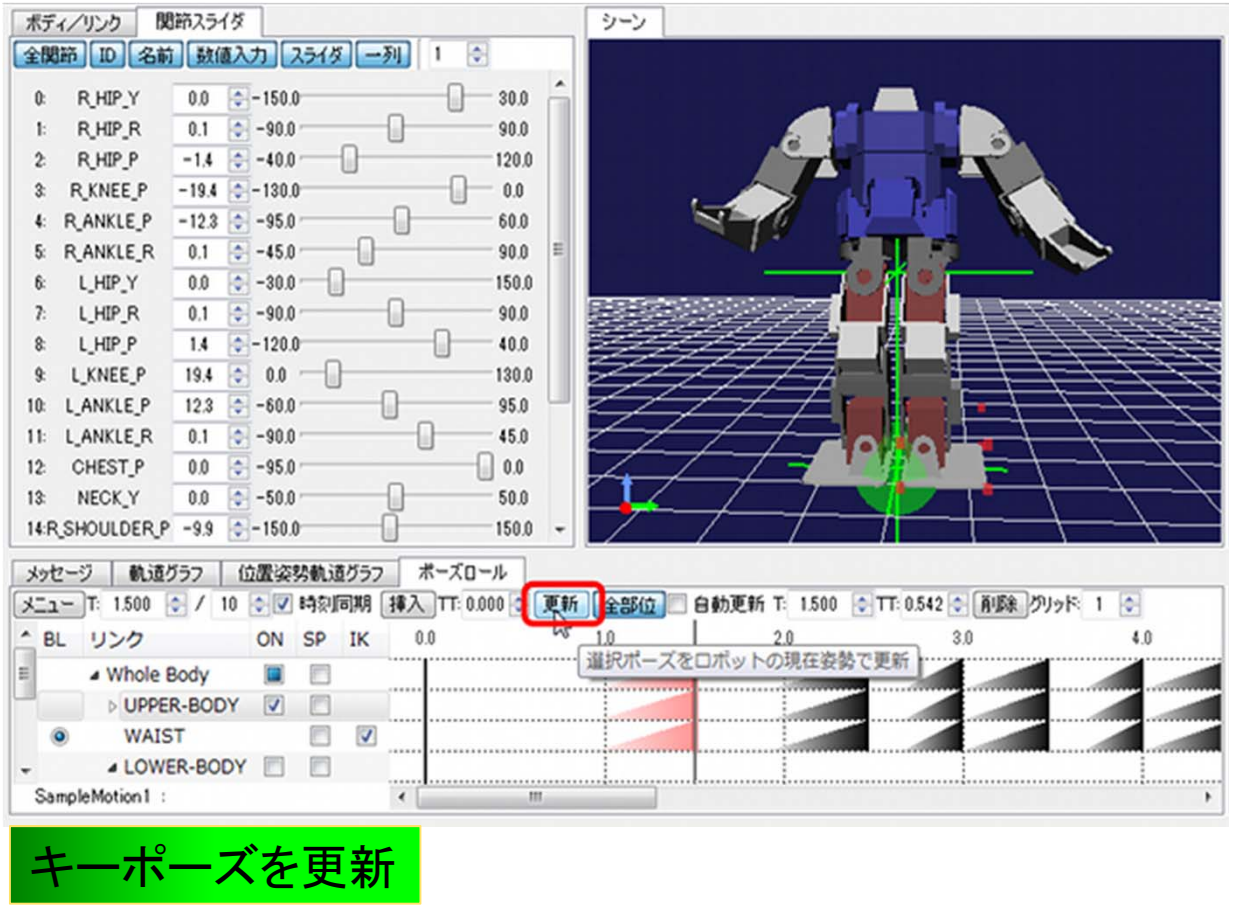
キーポーズの変更と追加



キーポーズの選択

- 動作パターンの編集は、キーポーズの追加と変更の繰り返し

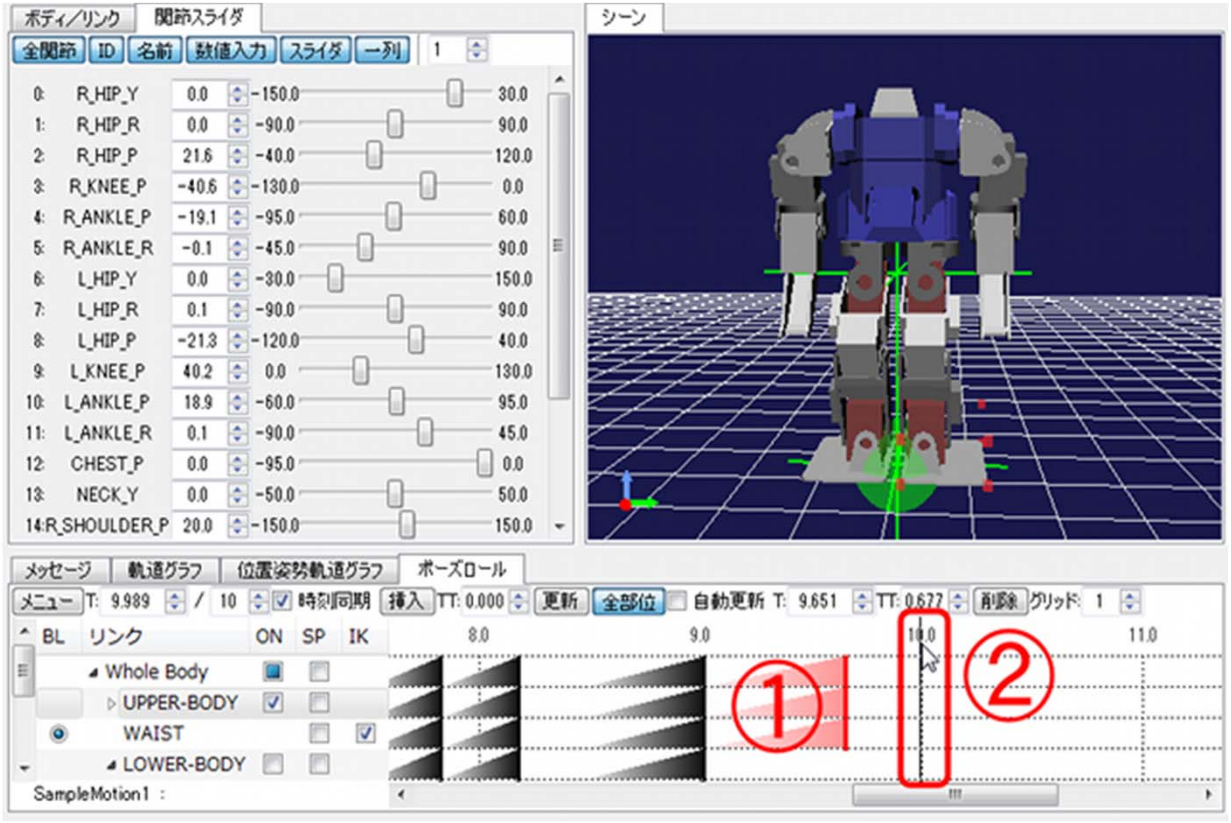
キーポーズの変更と追加



キーポーズを更新

- 動作パターンの編集は、キーポーズの追加と変更の繰り返し

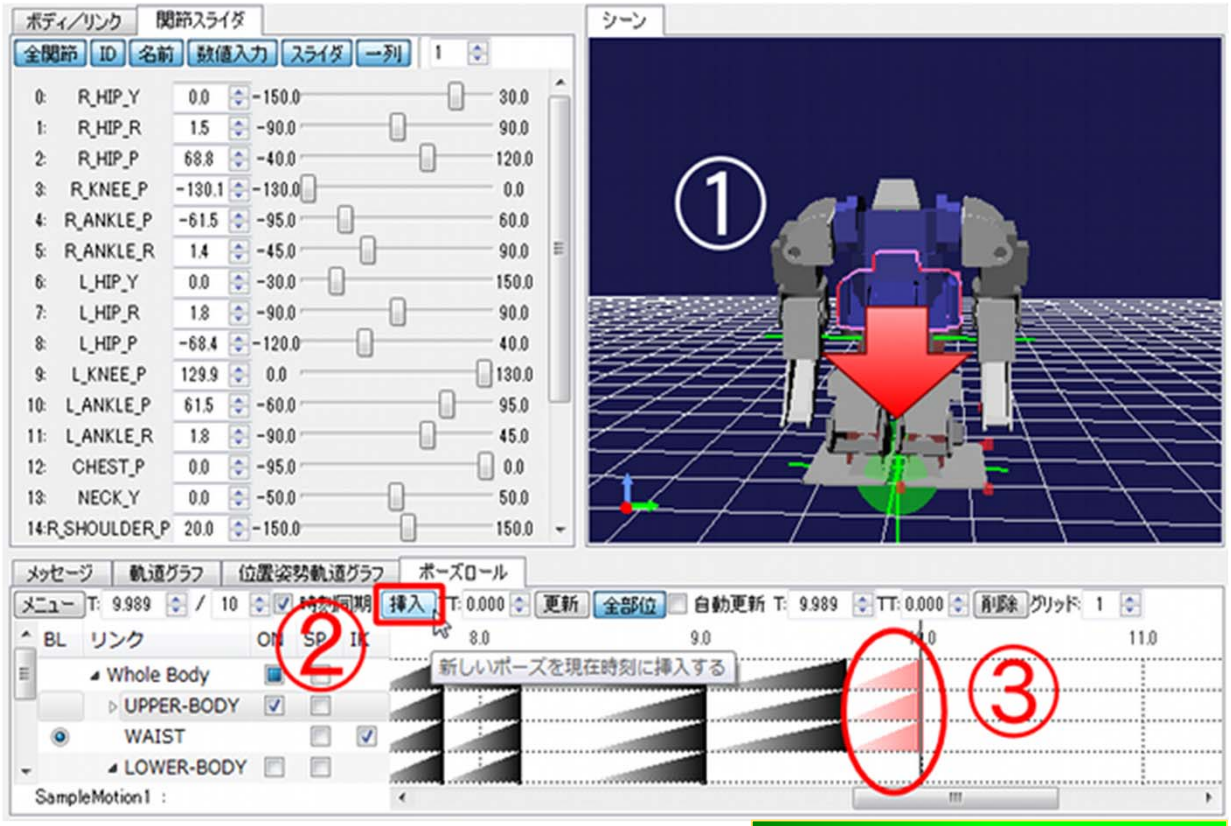
キーポーズの変更と追加



1．動作の末尾に移動
2．キーポーズの挿入時刻を指定

- 動作パターンの編集は、キーポーズの追加と変更の繰り返し

キーポーズの変更と追加

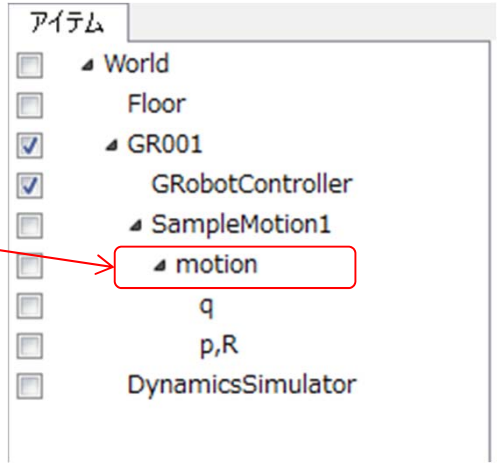


- 動作パターンの編集は、キーポーズの追加と変更の繰り返し

GR001のオリジナルの動作パターンの作成

- 作成した動作パターンからボディモーションパターンを生成（下図①）

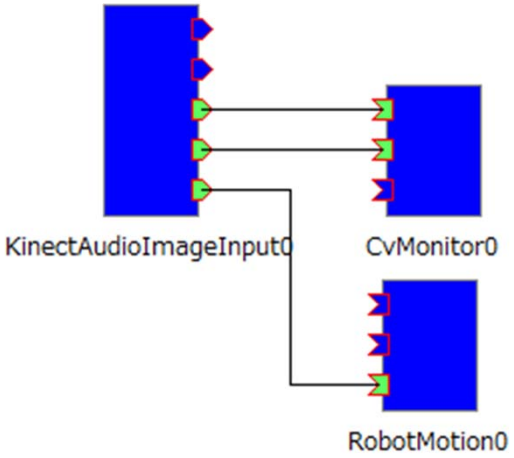
- アイテムを選択してボディモーションを保存
（C:¥Seminar-OpenHRI¥Choreonoid-1.1¥share¥
motions　に保存すること！！）



- 音声コマンド追加の要領で、新しい動作を音声コマンドで動作させる

KINECTを使ってG-ROBOT GR001を操作する

- KINECTをモーションキャプチャデバイスとして利用する。
- 使うコンポーネントetc.
  - KinectAudioImageInputRTC
    - KINECTがない人は、portaudioinputを利用する
  - Choreonoid
    - GR001.cnoid
    - RobotMotioRtcを新規作成
  - CvMonitor
    - KINECTからの出力のモニタ用



その他のコンポーネント

- GROBOT_vs10の下のファイル群
  - GROBOT_vs10/binの下
    - CommandInComp.exe
      - コマンドラインの命令入力用
    - GRobotRTCComp.exe
      - Choreonoidの動作ファイルを実行するためのRTC。GR001.dllを利用
    - GR001_Sample.exe
      - GR001.dllを利用したコマンド入力サンプル

- これらで作れるシステム
  - 2つのG-ROBOTに同じ動きをさせる
  - G-ROBOTでG-ROBOTを動かす
  - G-ROBOTでChoreonoidのロボットを動かす

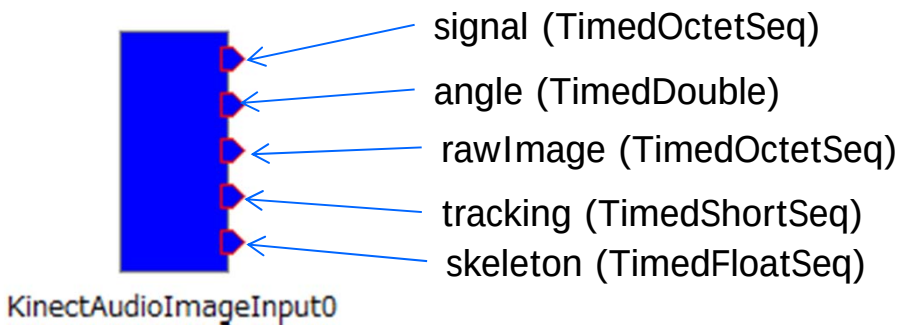
詳細は、
http://openrtp.jp/wiki/_default/ja/Software/GROBOTS.html

参考資料

- コンポーネントの概略
  - KinectAudioImageInput
  - CvMonitor
  - RobotoMotionRtc
  - JuliusRTC
  - SEAT

KinectAudioImageInput

本コンポーネントは、KINECTでキャプチャした音響データ、人物の姿勢データ、カラー画像を出力します。このコンポーネントで取得する姿勢データは、下記のようになっています。

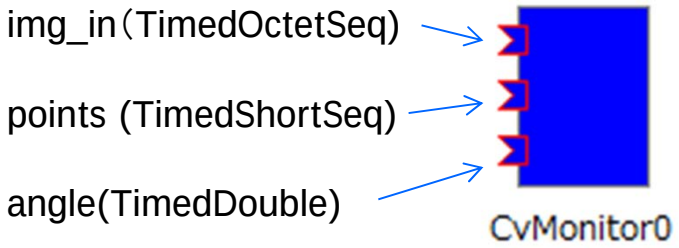



```
static int SkeletonPos[] = {
  NUI_SKELETON_POSITION_HEAD,
  NUI_SKELETON_POSITION_SHOULDER_LEFT,
  NUI_SKELETON_POSITION_ELBOW_LEFT,
  NUI_SKELETON_POSITION_WRIST_LEFT,
  NUI_SKELETON_POSITION_SHOULDER_RIGHT ,
  NUI_SKELETON_POSITION_ELBOW_RIGHT,
  NUI_SKELETON_POSITION_WRIST_RIGHT,
  NUI_SKELETON_POSITION_SPINE,
  NUI_SKELETON_POSITION_HAND_LEFT,
  NUI_SKELETON_POSITION_HAND_RIGHT,
  NUI_SKELETON_POSITION_HIP_CENTER,
  NUI_SKELETON_POSITION_HIP_LEFT,
  NUI_SKELETON_POSITION_HIP_RIGHT,
   };
```

名前	フローポート	データ型	説明
signal	OutPort	TimedOctetSeq	音声データ
angle	OutPort	TimedDouble	音源の方向（Rad）
rawImage	OutPort	TimedOctetSeq	カラー画像または深度画像（Configurationで設定）
tracking	OutPort	TimedShortSeq	姿勢データ（表示用）
skeleton	OutPort	TimedFloatSeq	姿勢データ（ロボット操作用）

CvMonitor

本コンポーネントは、KINECTコンポーネントからの出力を表示するためのものです。

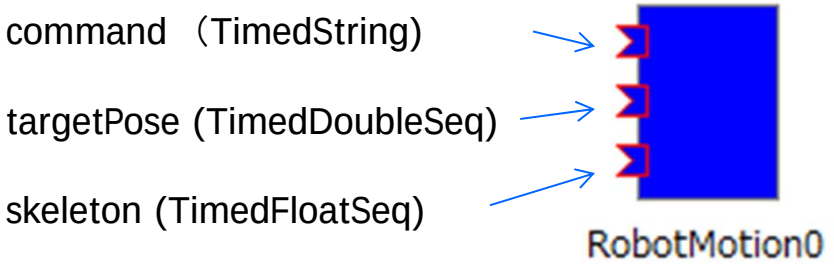


名前	フローポート	データ型	説明
img_in	InPort	TimedOctetSeq	Raw画像データ
points	InPort	TimedShortSeq	KINECTで取得した人の姿勢データ（表示用）
angle	InPort	TimedDouble	音源の方向（角度）。未実装

RobotMotionRtc

本コンポーネントは、ChoreonoidのRobotMotionItemによって生成されるRTコンポーネントです。動作パターン名、G-ROBOTの姿勢データ、KINECTで取得した人の姿勢データにもとづいて、Chorenoid内のロボットモデルの姿勢を変更します。

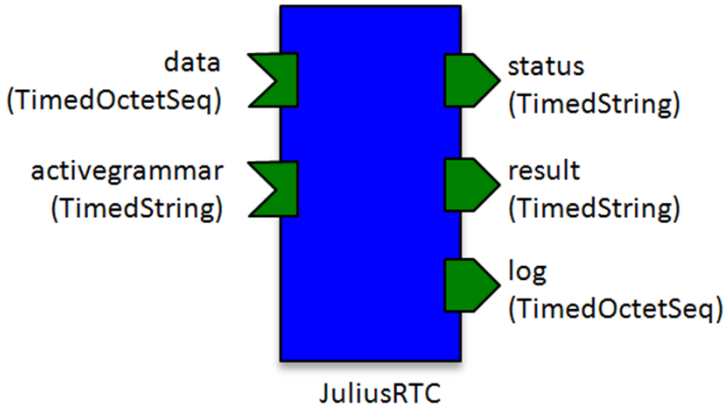
targetPoseと接続されるデータポートは、拙作のGRobotRTCから出力されるデータになります。



名前	フローポート	データ型	説明
command	InPort	TimedString	コマンド（動作パターン名）
targetPose	InPort	TimedDoubleSeq	G-ROBOTの各関節の角度データ
skeleton	InPort	TimedFloatSeq	KINECTで取得した人の姿勢データ

JuliusRTC

Juliusは、音声認識システムの開発・研究のためのオープンソースの高性能な汎用大語彙連続音声認識エンジンです。
本コンポーネントは、受け取った音声データをJuliusを用いて音声認識して認識テキストに変換します。

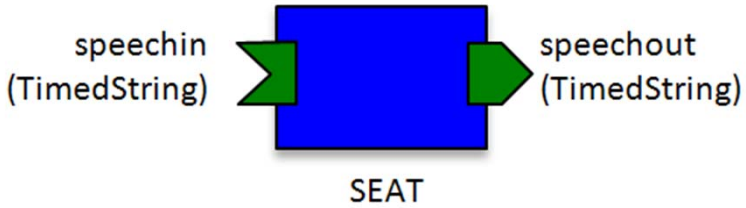


名前	フローポート	データ型	説明
data	InPort	TimedOctetSeq	認識する音声データ(パケット形式)
activegrammar	InPort	TimedString	有効化する文法ID
status	OutPort	TimedString	音声認識器の状態('LISTEN [音声入力受付中]', 'STARTREC [音声認識処理開始]', 'ENDREC [音声認識処理終了]', 'REJECTED [入力棄却]')
result	OutPort	TimedString	音声認識結果(XML形式)
log	OutPort	TimedOctetSeq	音声データのログ

SEAT

SEAT(Speech Event Action Transfer)は、音声対話制御を実現するためのソフトウェアです

```
<?xml version="1.0" encoding="UTF-8"?>
  <seatml>
    <general name="sample">
      <agent name="speechin" type="rtcin" datatype="TimedString" />
      <agent name="speechout" type="rtcout" datatype="TimedString" />
    </general>
</seatml>
```



名前	フローポート	データ型	説明
speechin	InPort	TimedString	音声認識結果のテキスト
speechout	OutPort	TimedString	発話するテキスト