ChoreonoidとOpenHRIを用いた対話型ロボットのRTC群について

（独）産業技術総合研究所
知能システム研究部門
原　　功

講義内容

- Choreonoidの概要
  - Choreonoidの開発背景
  - Choreonoidのシステム構成
  - Choreonoidの使い方
- OpenHRIの概要
  - OpenHRIの開発背景
  - OpenHRIで提供されている機能
- ChoreonoidとOpenHRIを用いたロボティクスの研究開発事例
  - HRP-4C
  - GraspPlugin for Choreonoid
  - OpenHRIを使ったコミュニケーションロボット

講義内容

- Choreonoidの概要
  - Choreonoidの開発背景
  - Choreonoidのシステム構成
  - Choreonoidの使い方
- OpenHRIの概要
  - OpenHRIの開発背景
  - OpenHRIで提供されている機能
- ChoreonoidとOpenHRIを用いたロボティクスの研究開発事例
  - HRP-4C
  - GraspPlugin for Choreonoid
  - OpenHRIを使ったコミュニケーションロボット

Choreonidの開発背景

- 1996年末　ホンダヒューマノイド　P2発表
- 1998年～2003年 「人間協調・共存型ロボットシステムの研究開発」プロジェクト

以降、さまざまな人型のロボットが登場

→ ロボットの動作教示が複雑化

→ 各関節角の目標角度を直接入力では限界がある

→ モーションキャプチャの利用 → 高価な機器が必要

OpenHRP3などのシミュレータを利用

→ 動作が遅く不安定

→ CGを作るようにもっと簡単に動作を作成できないか？

Choreonidの開発背景

- OpenHRP3のように動力学シミュレーションが実行でき、CG製作者でも簡単に扱えるツールを実現したい

- OpenHRP3の不満点
  - Java3Dを利用しているため、動作が遅い
  - メモリ容量の限界、ガベージコレクション
  - 動力学計算は、C++なのに…

知能化PJにおいて開発したロボット知能ソフトウェアプラットフォームの1つのツールとしてフルスクラッチでから開発　← UIはOpenHRP3との親和性を持たせることが条件で…

基本的な設計方針

- 内部計算処理だけでなく、3Dレンダリングを含む可視化、アニメーション、およびユーザ入力と内部計算との連携も含めて、コンピュータの性能を最大限活用可能な設計とする。
- 必要に応じてユーザが機能を柔軟に拡張可能とする。
- ロボットや計算機の専門家ではないユーザにも使いやすいツールとする。

MVCモデルに基づく実装

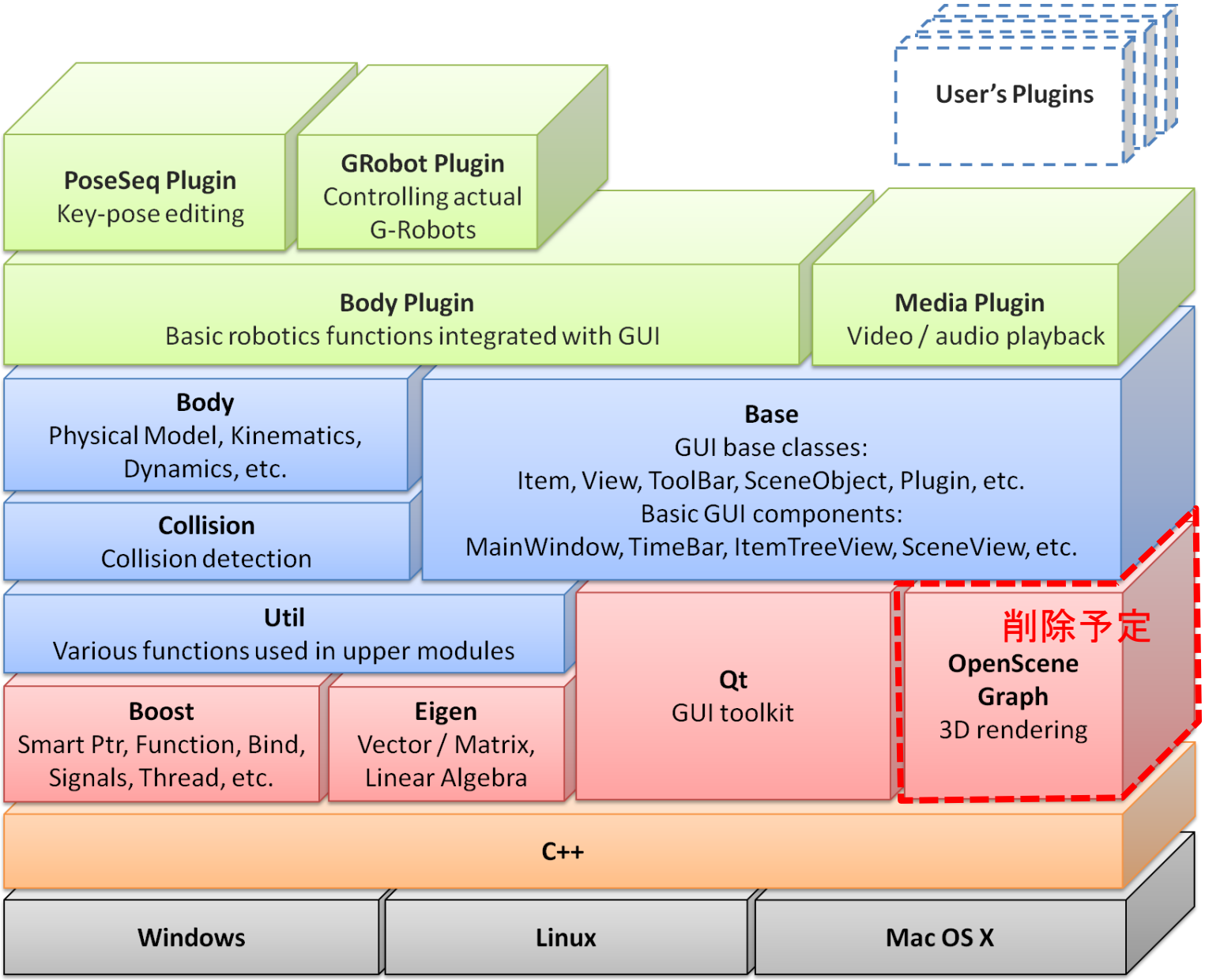
→ロボットアプリケーションのフレームワークとしても利用可能

Choreonidの特徴

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

Choreonoidのシステム構成



Choreonoidの標準プラグイン

- **BodyPlugin:**ロボットモデルや動作データに関するGUIや処理を実装したプラグイン

- **PoseSeqPlugin:** ロボットのキーフレーム編集を行う機能を実装したプラグイン

- **GRobotPlugin:** HPIジャパン株式会社の小型ロボット”G-Robot”の実機をモデルの動作と連動させて動かすための機能を実装したプラグイン

- **MediaPlugin:** ビデオや音声をロボットの動きと合わせて再生するためのプラグイン

Choreonoidの機能拡張

Choreonoid は、プラグインの追加によって…

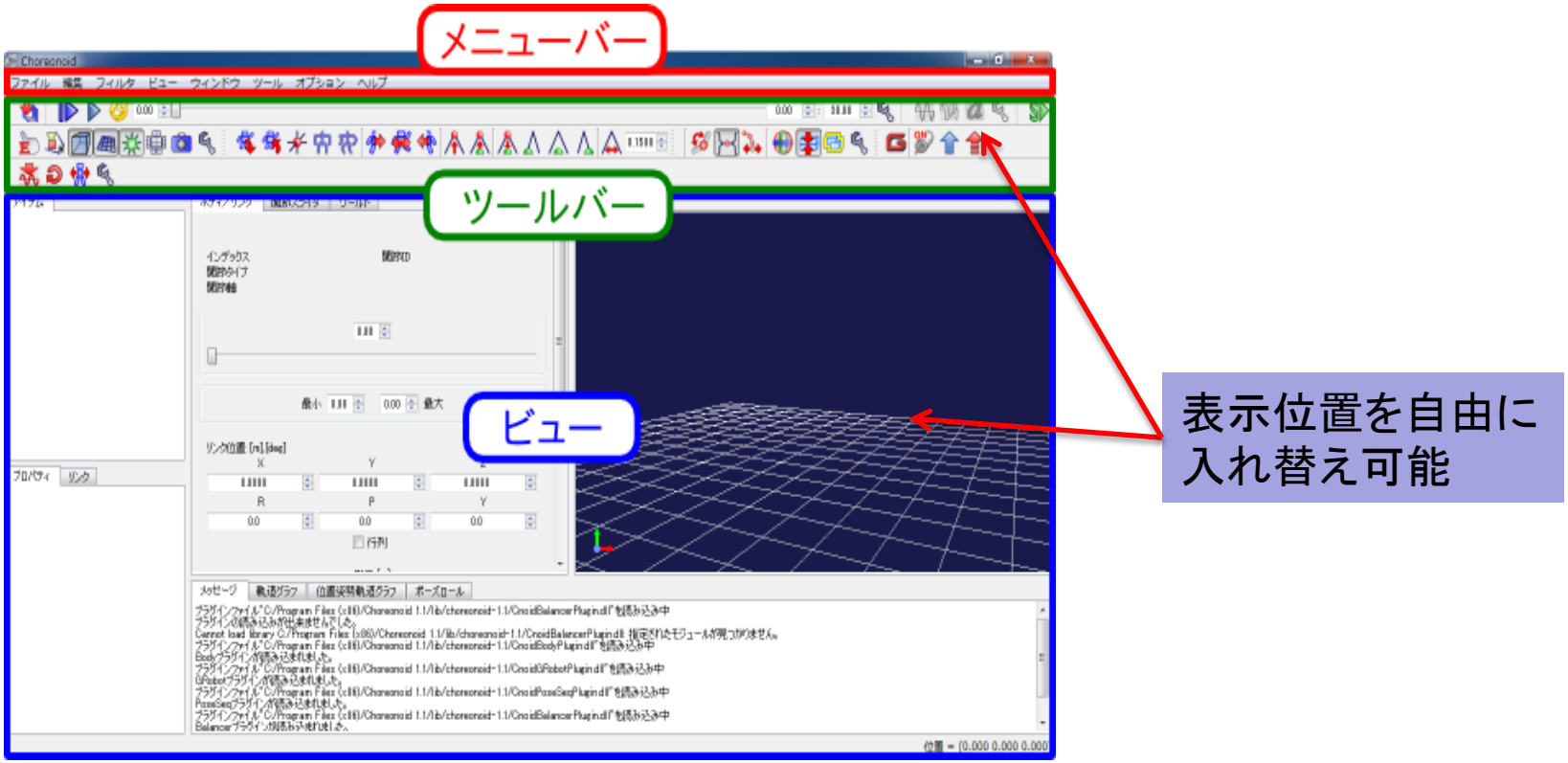
- 動作計画ツールとして
- ロボット操作IFとして
- ロボットシミュレータとして
- シナリオ記述するツールとして
- モデルデザインツールとして

利用することができる

Chreonoidを自分のロボットで利用する

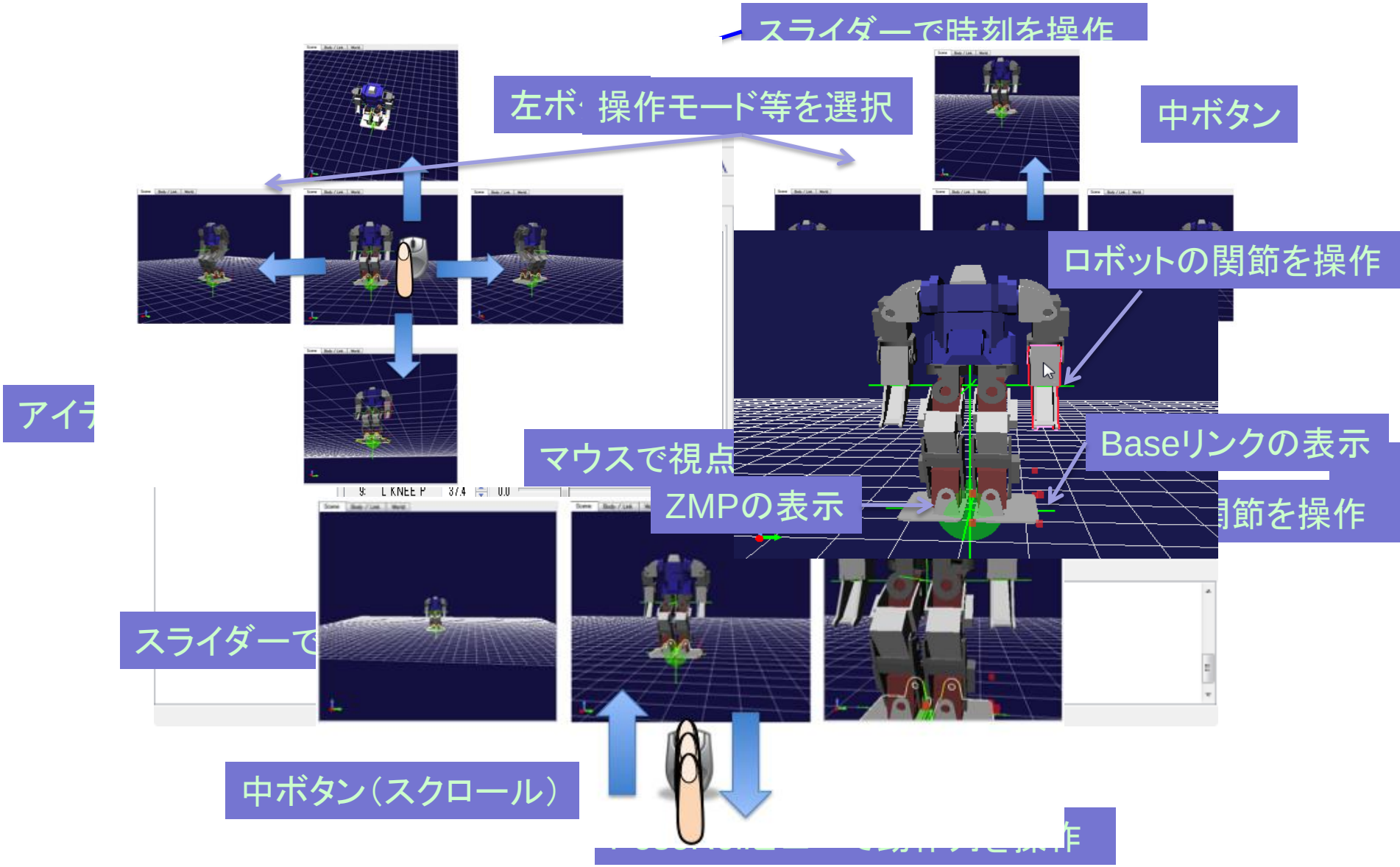
- Chreonoidを利用するに必要なもの
  - 対象となるロボットのモデル
    - ロボットのパーツのVRMLモデル
    - 各パーツの重心位置と慣性モーメントマトリックス
  - IK計算用プログラムモジュール
    - 独自開発
    - OpenRAVE提供のikfastを利用する
  - ロボット制御ソフトへ命令を与えるためのプラグイン

Choreonoidの基本的な操作

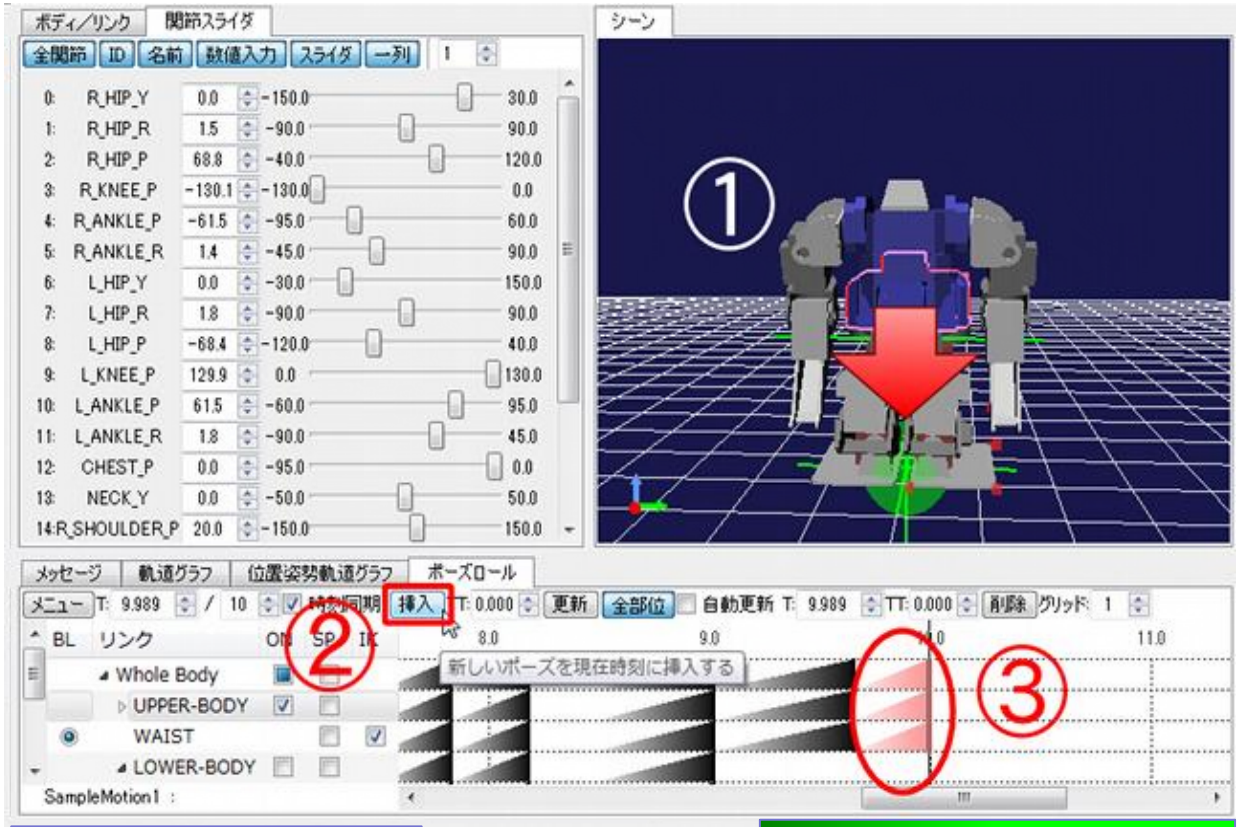


- メニューバー
  - 本メニューバーに格納されているメニューを用いることで、Choreonoidの各種操作を行うことが出来ます。メニュー項目はプラグインによって追加することも可能です。
- ツールバー
  - ツールバー領域には、ボタンやスライダ、数値入力ボックス等のGUI部品で構成されるツールバーが配置されます。ツールバーは機能ごとにグループ化されたものとなっており、各ツールバーの左端をマウスでドラッグすることで簡単に好みの場所に移動させることができます。
- ビュー
  - ビューは、Choreonoidにおいて各種情報を表示・編集するためのウィンドウ領域です。タブ内に格納される各矩形領域がひとつのビューに対応します。
  - Choreonoid本体に備わった基本的なビューとして、各種データ・モデル等（アイテム）を管理する「アイテムビュー」、各アイテムのプロパティを表示・編集するための「プロパティビュー」、3D表示にてモデルの確認や編集を行うための「シーンビュー」、テキストメッセージが出力される「メッセージビュー」などがあります。

Choreonoidの基本的な操作



キーポーズの変更と追加



キーポーズを更新

1. 動作の
2. キーポー

1. キーポーズを生成
2. キーポースを挿入
3. 動作時間を調整

- 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

Choreonoidにおけるロボットの動作表現

- Choreonoidでは、ロボットの動きをキーポーズの連続として表現する

- キーポーズは、動作中の状態が変化するポイントの姿勢

- 無理な姿勢のキーポーズは、身体バランス補正を行い、安定動作を生成する。

- ロボットへは、制御時間ごとの目標姿勢に変換して、命令列を与えて実行させる

Choreonoidで生成される動作パターンファイル

Choreonoidでは、YAML形式のファイルを使用

- Pose Sequenceファイル(.pseq)
  - 時刻、動作時間、動作する関節角の目標角度、IK計算のための情報
- Motionファイル(.yaml)
  - Pose Sequenceから生成される
  - 一定時間間隔の関節角の列、関節重心位置、ZMPの位置などの情報

SampleMotion1.pseq

```
type: PoseSeq
name: "SampleMotion1"
targetBody: "GR001"
refs:
 -
  time: 0
  refer:
   type: Pose
   name: ""
   joints: [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15, 16, 17, 18, 19 ]
   q: [
    -1.80616593e-19, 0.00197016157, 0.370920524, -0.701176708, -0.330256184, 0.00197016157,
    -6.05607271e-19, 0.00197016157, -0.370920524, 0.701176708, 0.330256184, 0.00197016157,
    0.34906585, 0, -0.34906585, -0.34906585, 0, 0.34906585 ]
   ikLinks:
    -
     name: WAIST
     index: 0
     isBaseLink: true
     translation: [ -0.00206589105, 0.000206960048, 0.154197111 ]
     rotation: [
      1, 7.25663958e-19, 7.30074269e-16,
      -4.21994409e-19, 1, 3.56094663e-15,
      -5.75440303e-16, -3.56007164e-15, 1 ]
    -
     ...
```

データのタイプ

対象となる関節のID

IKに関する情報

目標関節角

SampleMotion1.yaml

```
# Body motion data set format version 1.0 defined by cnoid-Robotics

type: BodyMotion
components:
 -
  type: "MultiValueSeq"
  purpose: "JointPosition"
  frameRate: 50
  numParts: 20
  numFrames: 534
  frames:
   - [ 5.67373576e-011, 0.00197014097, 0.37092773, -0.701189657, -0.330261928,
       0.00197014107, -8.17846501e-011, 0.00197018217, -0.370927729, 0.701189655,
       0.330261927, 0.00197018206, 0, 0, 0.34906585, 0, -0.34906585, -0.34906585, 0,
       0.34906585 ]

   - [ 5.67373566e-011, 0.00197014105, 0.370927731, -0.701189657, -0.330261927,
       0.00197014115, -8.17846502e-011, 0.00197018225, -0.37092773, 0.701189655,
       0.330261926, 0.00197018214, 0, 0, 0.34906585, 0, -0.34906585, -0.34906585, 0,
       0.34906585 ]

                ...
```

データのタイプ

更新頻度

関節数

目標関節角

Choreonoidの機能拡張

- Choreonoid プラグイン
  - Choreonoid Plugin を追加することで、Choreonoidに様々な機能を追加することができる
  - GUIを伴ったロボットソフトウェア開発、操作環境を簡単に構築

•Boost Signalsライブラリによるイベント処理
•Boost Bindライブラリによるコールバック関数の自動生成

プラグイン開発の流れ

- Choreonidフレームワークのヘッダをインクルード
- プラグインクラスの定義
  - Cnoid::Plugin　のクラスを継承して定義
  - コンストラクタには、プラグイン間の依存関係を’require’ 関数で通知
  - Initialize関数の定義
    - プラグインを初期化、メニュー、ツールバー等の定義
    - 成功すればtrueを返す
  - プラグインの実行関数の定義
- プラグインエントリの定義

HelloWorldPlugin

```
#include <cnoid/Plugin>
#include <cnoid/MenuManager>
#include <cnoid/MessageView>
#include <boost/bind.hpp>

using namespace cnoid;
using namespace boost;

class HelloWorldPlugin : public Plugin
{
public:

    HelloWorldPlugin() : Plugin("HelloWorld")
    {

    }

    virtual bool initialize()
    {
        Action* menuItem = menuManager().setPath("/View").addItem("Hello World");
        menuItem->sigTriggered().connect(bind(&HelloWorldPlugin::onHelloWorldTriggered, this));
        return true;
    }

private:

    void onHelloWorldTriggered()
    {
        MessageView::mainInstance()->putln("Hello World !");
    }
};

CNOID_IMPLEMENT_PLUGIN_ENTRY(HelloWorldPlugin)
```

ヘッダのインクルード

プラグインクラスの定義

メニューの定義

メニューの実行関数

プラグインエントリの定義

- メニューを選択するとメセージを表示する。

HelloWorldPlugin(メニューの作成)

- MenuManagerの取得

```
MenuManager mMgr = menuManager().setPath("/View");
```

- メニューItemの追加

```
Action* menuItem = mMgr.addItem("Hello World");
```

- SignalProxyの取得

```
SignaleProxy<boost::signal<void(void)>> handle
        = menuItem->sigTriggered();
```

- メニューへ関数を結びつける

```
handle.connect(bind(&HelloWorldPlugin::onHelloWorldTriggered, this));
```

- メンバー関数を汎用関数オブジェクトに変換する

```
boost::function<void(void)> func
    =bind(&HelloWorldPlugin::onHelloWorldTriggered, this);
```

SamplePlugin

```
#include <cnoid/Plugin>
#include <cnoid/ItemTreeView>
#include <cnoid/BodyItem>
#include <cnoid/ToolBar>
#include <boost/bind.hpp>

using namespace boost;
using namespace cnoid;

class Sample1Plugin : public Plugin
{
public:

  Sample1Plugin() : Plugin("Sample1")
  {
    require("Body");
  }

  virtual bool initialize()
  {
    ToolBar* bar = new ToolBar("Sample1");
    bar->addButton("Increment")
       ->sigClicked().connect(bind(&Sample1Plugin::onButtonClicked, this, +0.04));
    bar->addButton("Decrement")
       ->sigClicked().connect(bind(&Sample1Plugin::onButtonClicked, this, -0.04));
    addToolBar(bar);
    return true;
  }

  void onButtonClicked(double dq)
  {
    ItemList<BodyItem> bodyItems =
       ItemTreeView::mainInstance()->selectedItems<BodyItem>();

    for(size_t i=0; i < bodyItems.size(); ++i){
      BodyPtr body = bodyItems[i]->body();
      for(int j=0; j < body->numJoints(); ++j){
        body->joint(j)->q += dq;
      }
      bodyItems[i]->notifyKinematicStateChange(true);
    }
  }
};

CNOID_IMPLEMENT_PLUGIN_ENTRY(Sample1Plugin)
```

ヘッダのインクルード

プラグインクラスの定義

プラグインの依存関係の通知

ツールバーの定義

BodyItemの取得

ツールバーの実行関数

状態変更の通知

プラグインエントリの定義

- 選択されているロボットの姿勢を変更するプラグイン
- ツールバーで操作

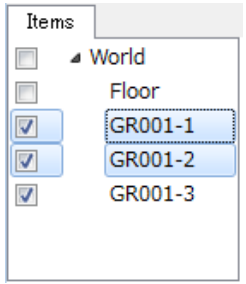
SamplePlugin

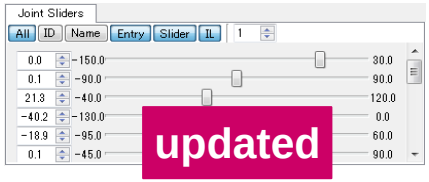
```
class SamplePlugin : public Plugin {
public:
    SamplePlugin() : Plugin("Sample") { require("Body"); }
    virtual bool initialize() {
        ToolBar* bar = new ToolBar("Sample1");
        bar->addButton("Increment ")
            ->sigClicked().connect(bind(&SamplePlugin::onButtonClicked, this, +0.04));
        bar->addButton("Decrement ")
            ->sigClicked().connect(bind(&SamplePlugin::onButtonClicked, this, -0.04));
        addToolBar(bar);
        return true;
    }
    void onButtonClicked(double dq){
      ItemList<BodyItem> bodyItems =
        ItemTreeView::mainInstance()->selectedItems<BodyItem>();
      for(size_t i=0; i < bodyItems.size(); ++i){
        BodyPtr body = bodyItems[i]->body();
        for(int j=0; j < body->numJoints(); ++j){
            body->joint(j)->q += dq;
        }
        bodyItems[i]->notifyKinematicStateChange(true);
      }
    }
```

Increment

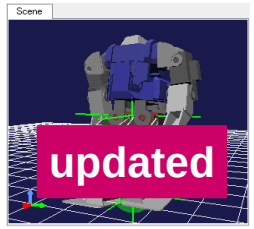
Decrement

clicked





signal



signal

SamplePlugin(ツールバーの定義)

- ツールバー生成

```
ToolBar* bar = new ToolBar("Sample1")
```

- ボタン生成

```
ToolButton *button = bar->addButton("Increment");
```

- SignalProxyの取得

```
SignalProxy<boost::signal<void(bool)>> click_func = button->sigClicked();
```

- クリック時に呼ばれる関数の結びつけ

```
click_func.connect(bind(&Sample1Plugin::onButtonClicked, this, +0.04));
```

- メンバー関数を汎用関数オブジェクトに変換する

```
boost::function<void(void)> func
    = bind(&Sample1Plugin::onButtonClicked, this, +0.04)
```

SamplePlugin(BodyItemの操作)

- ItemViewから選択されたItemの取得

```
ItemList<BodyItem> bodyItems =
        ItemTreeView::mainInstance()->selectedItems<BodyItem>();
```

- 各BodyItemに対して操作する

```
for(size_t i=0; i < bodyItems.size(); ++i){

    //  スマートポインタに変換する。
    //  typedef boost::shared_ptr<Body> BodyPtr
    BodyPtr body = bodyItems[i]->body();

    //  Bodyオブジェクトに対して処理を実行
    for(int j=0; j < body->numJoints(); ++j){
      body->joint(j)->q += dq;
    }

   // モデル全体とGUIに状態変更を通知
     bodyItems[i]->notifyKinematicStateChange(true);

}
```

プラグインのコンパイルとインストール

- プラグインのコンパイル
  - インストール済みのChoreonoidを使う
    - Choreonoid依存ライブラリ、ヘッダーのパスを適切に設定する必要がある
  - Choreonoid本体のコンパイル環境を使う
    - Choreonoidのソースツリーのextpluginの下にコピーして一括コンパイル
    - CMakeList.txtを書く必要がある
- プラグインのインストール
  - {ChoreonoidのTopDir}/lib/choreonoid-1.1の下にDLLをコピーする

Choreonoidの次期バージョン（予定）

- 動力学シミュレーションプラグイン
  - AISTの動力学エンジン（OpenHRP3のもの）
  - ODE (OpenDynamics Engine)
- CORBAへの対応
  - ネームサーバー
  - OpenRTM
- 非CORBA通信コントローラサンプル
- 2Dシミュレーション

講義内容

- Choreonoidの概要
  - Choreonoidの開発背景
  - Choreonoidのシステム構成
  - Choreonoidの使い方
- OpenHRIの概要
  - OpenHRIの開発背景
  - OpenHRIで提供されている機能
- ChoreonoidとOpenHRIを用いたロボティクスの研究開発事例
  - HRP-4C
  - GraspPlugin for Choreonoid
  - OpenHRIを使ったコミュニケーションロボット

OpenHRIの開発背景

- 知能PJのコミュニケーション知能の開発で、コンポーネントの互換性がほしい
- ATR、NEC、産総研を中心に、音声認識、音声発話の標準IFを作成
- 標準IFに基づきフリーで利用できる各オープンソースソフトウェアでコミュニケーション知能のモジュール群を作成する

OpenHRIとは

- OpenHRIは、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

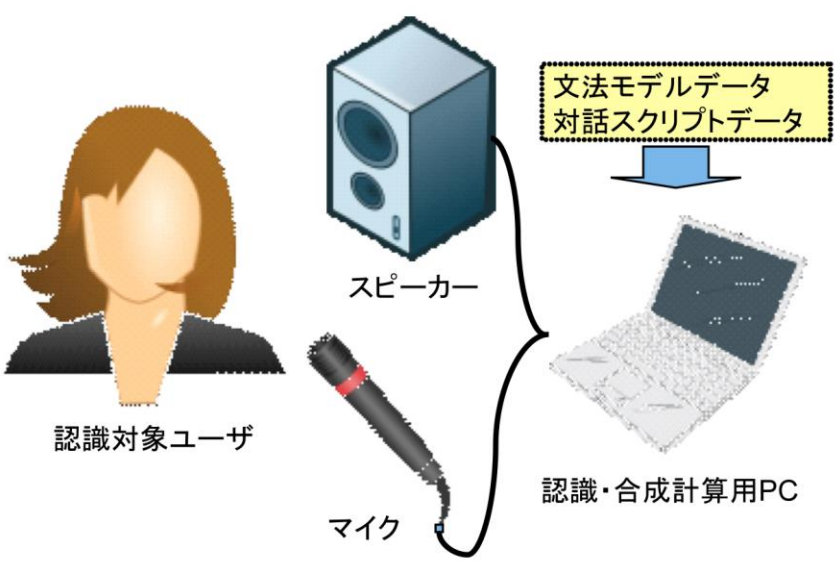
- フリーで利用できる各オープンソースソフトウェアを使い易いコンポーネントとしてまとめたもの

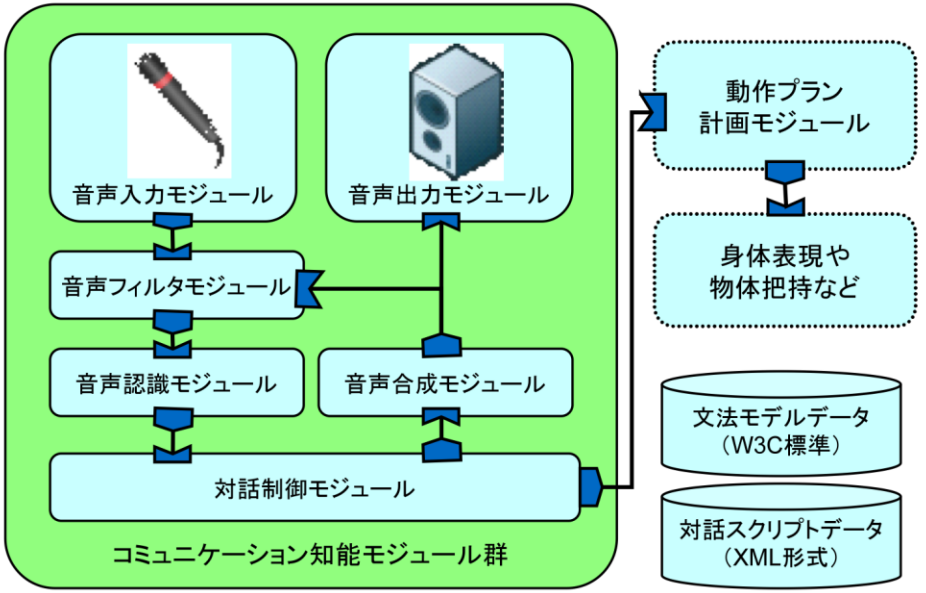
- 知能化PJにおいて開発され、EPL-1.0にて公開

  http://openhri.net

OpenHRIで提供する機能

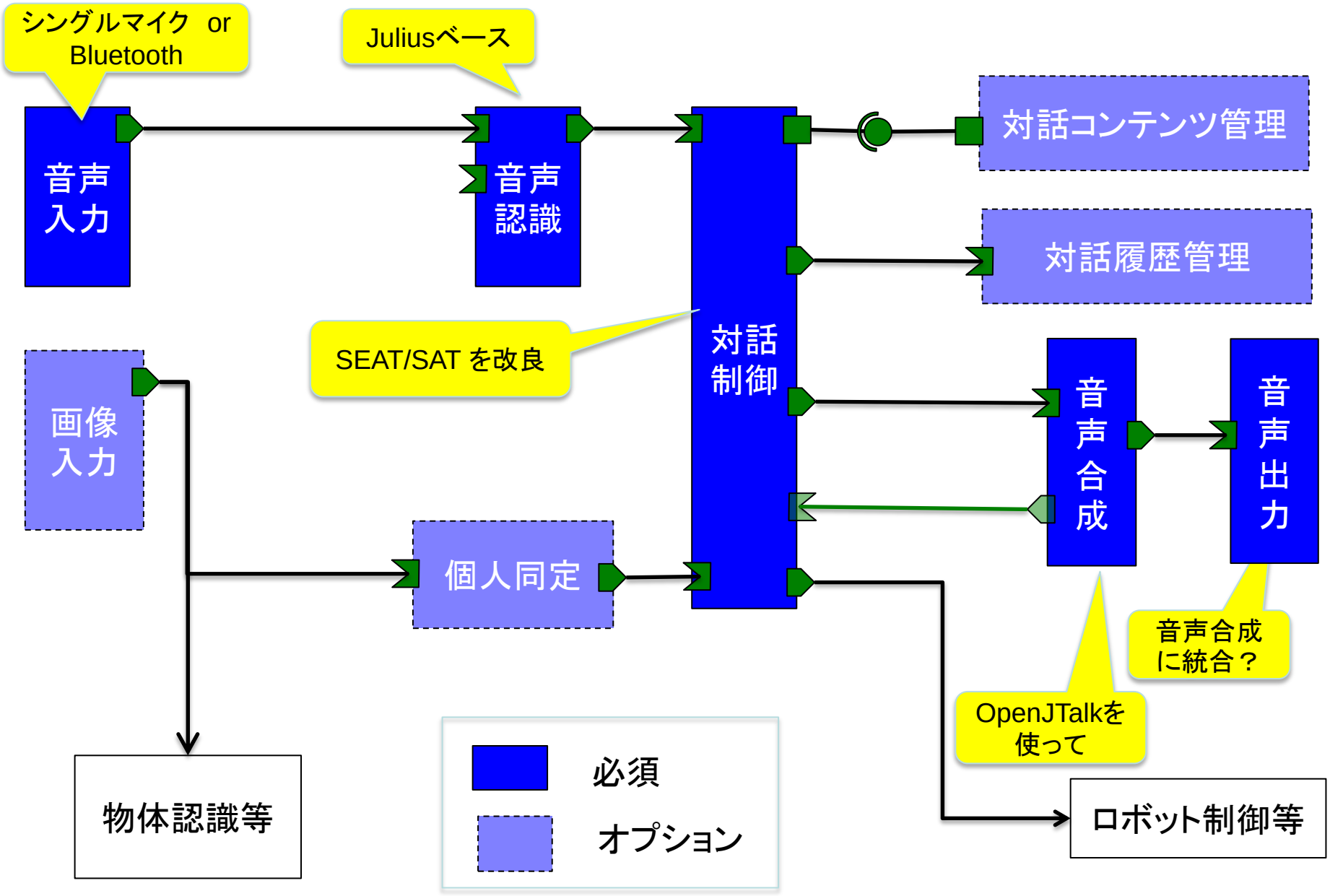
- ロボットのコミュニケーション機能の実現に必要な各要素
  （音声入出力・音声認識・音声合成・対話制御など）





※IFは全て各社共通形式（どのモジュールでも差し替え可能です）

OpenHRIで提供する機能



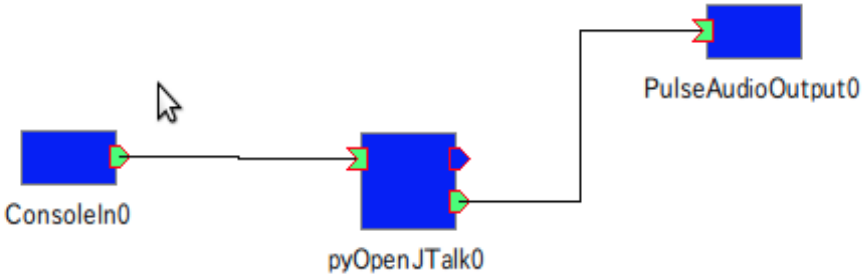
OpenHRIの利用環境

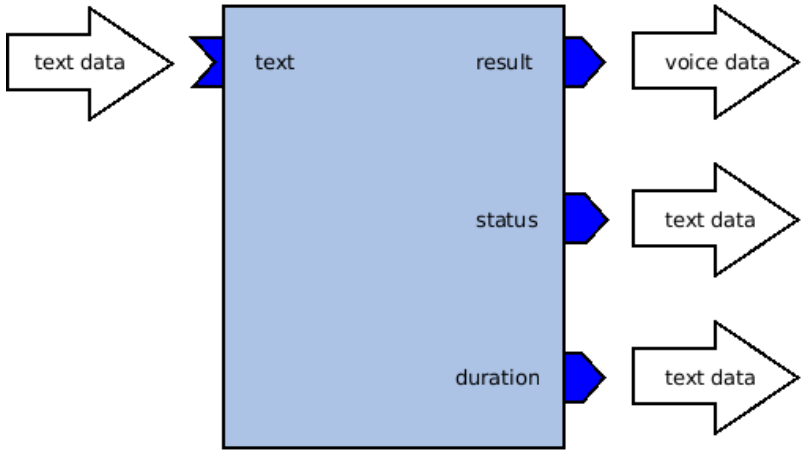
- バイナリパッケージ動作環境
  - WindowsXP以上またはUbuntu Linux 10.04においてOpenRTM-aist-1.0（Python)が動作し、RTSエディタ等が動作すること
  - Windows用インストーラとUbuntu Linux 10.04のSynapicパッケージ・マネージャに対応
  - 音声入力装置（マイクロホン等）および音声出力装置（スピーカ等）が必要

話す機能

- 音声合成コンポーネントは、フリーで利用できる音声合成エンジン(OpenJTalk、Festival) 、を利用
- テキストを入力とし、音声データ(Mono,16kHz, 16bit)を出力

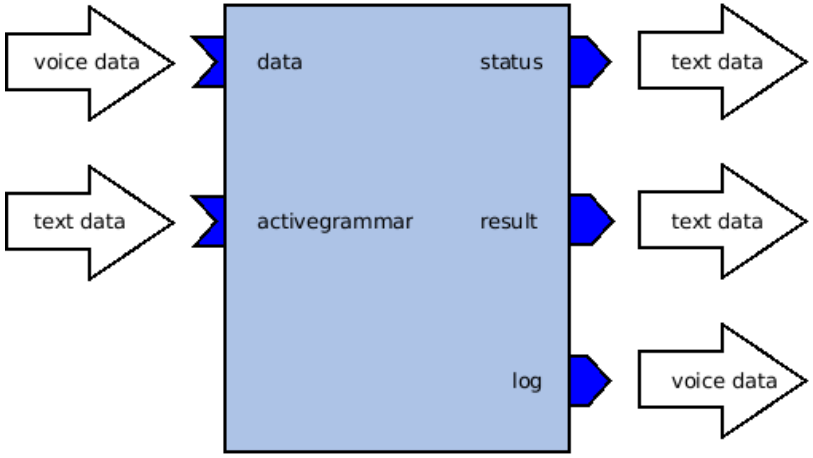
参考システム構成





聴く機能

- 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Juliusを使った音声認識機能
- 受け取った音声データ(16bits, 16KHz)を音声認識して認識テキストに変換する
- 音声認識は、予め用意された音声認識文法ファイルに従って実行される



聴く機能

- 音声認識文法
  - 音声認識文法は、音声認識器に与える認識可能な文法（単語の構造）を定義したもの

    W3C-Speech Recognition Grammar Specificationに準拠した記述を使用
    - 検証ツール： validatesrgs
    - 視覚化ツール： srgstojulius

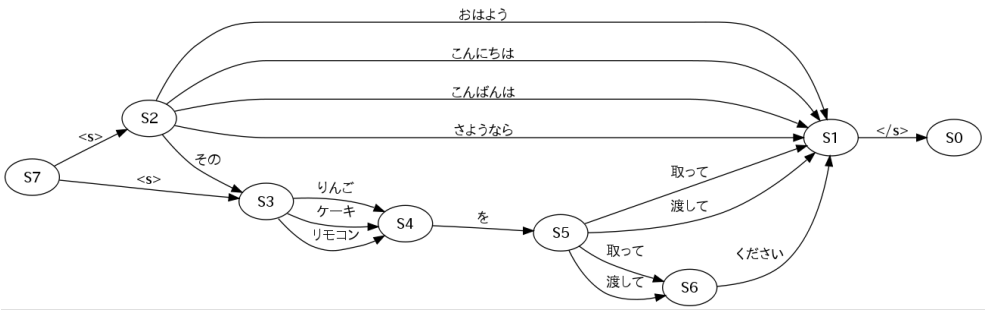
聴く機能

- 音声認識文法の作成
  - 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。

聴く機能

- 音声認識文法の例
  - 簡単な挨拶
  - 手渡し依頼

視覚化ツールによる出力




```
<?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="main">
 <lexicon uri="sample-lex-jp.xml"/>
 <rule id="main">
    <one-of>
        <item><ruleref uri="#greet" /></item>
       <item><ruleref uri="#command" /></item>
     </one-of>
 </rule>
 <rule id="greet">
     <one-of>
       <item>おはよう</item>
       <item>こんにちは</item>
       <item>こんばんは</item>
       <item>さようなら</item>
    </one-of>
 </rule>
 <rule id="command">
  <item repeat="0-1">その</item>
  <one-of>
   <item>りんご</item>
   <item>ケーキ</item>
  <item>リモコン</item>
 </one-of>
   <item>を</item>
  <one-of>
   <item>取って</item>
   <item>渡して</item>
 </one-of>
 <item repeat="0-1">ください</item>
 </rule>
 </grammar>
```

対話を制御する機能

- SEAT(Speech Event Action Transfer)とは？
  - ロボットとの対話を実現するための小さく軽量な対話エンジン
  - ロボットの対話動作は、音声認識結果（条件）とシステムの動作（アクション）のペアを基本として対話動作を記述
  - 状態遷移モデルに基づく対話管理
  - XMLベースのSEATMLにより対話スクリプトを記述

SEATML

SEATMLとは？

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。

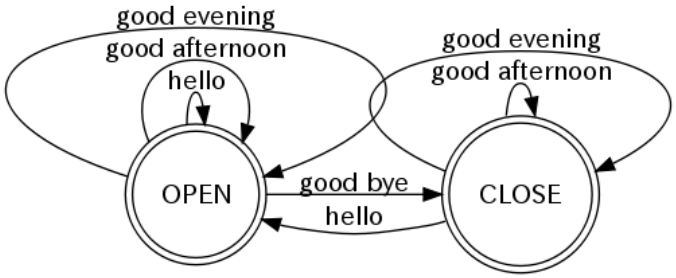
- アダプタの定義
  - SEATには、名前と通信方法(RTMとソケット)を対応付けるアダプタ機構を持っています。アダプタ機構は、通信方法の差異を隠蔽化することで、システムのハードウェア構成の変化に適応し、対話ロジックの再利用性を向上させます。
  - General : アダプタ定義部を示します。
  - Agent : 名前と通信方法の対応を示します。“type”属性は”rtcin”、”rtcout”、”socket”を取ることができます。タイプが”rtcin”か”rtcout”と定義されたとき、”datatype”属性を定義できます(データ型に関しては、RTMの仕様を参照してください)。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。

SEATML

```
<?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>
```

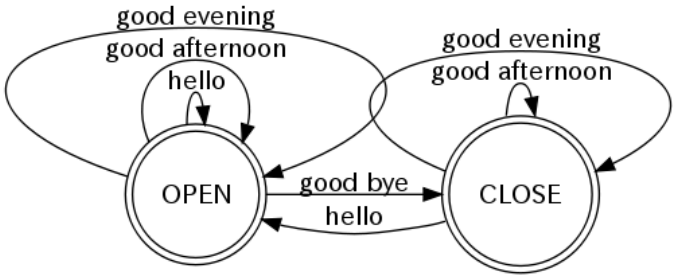


SEATMLサポートツール

OpenHRIでは、SEATMLを記述するために下記のツールを提供

- 検証ツール：validateseatml
  - SEATMLの文法チェックコマンド

- 視覚化ツール：seatmltographviz
  - SEATMLスクリプトをグラフ表示させて構造をチェックするツール



講義内容

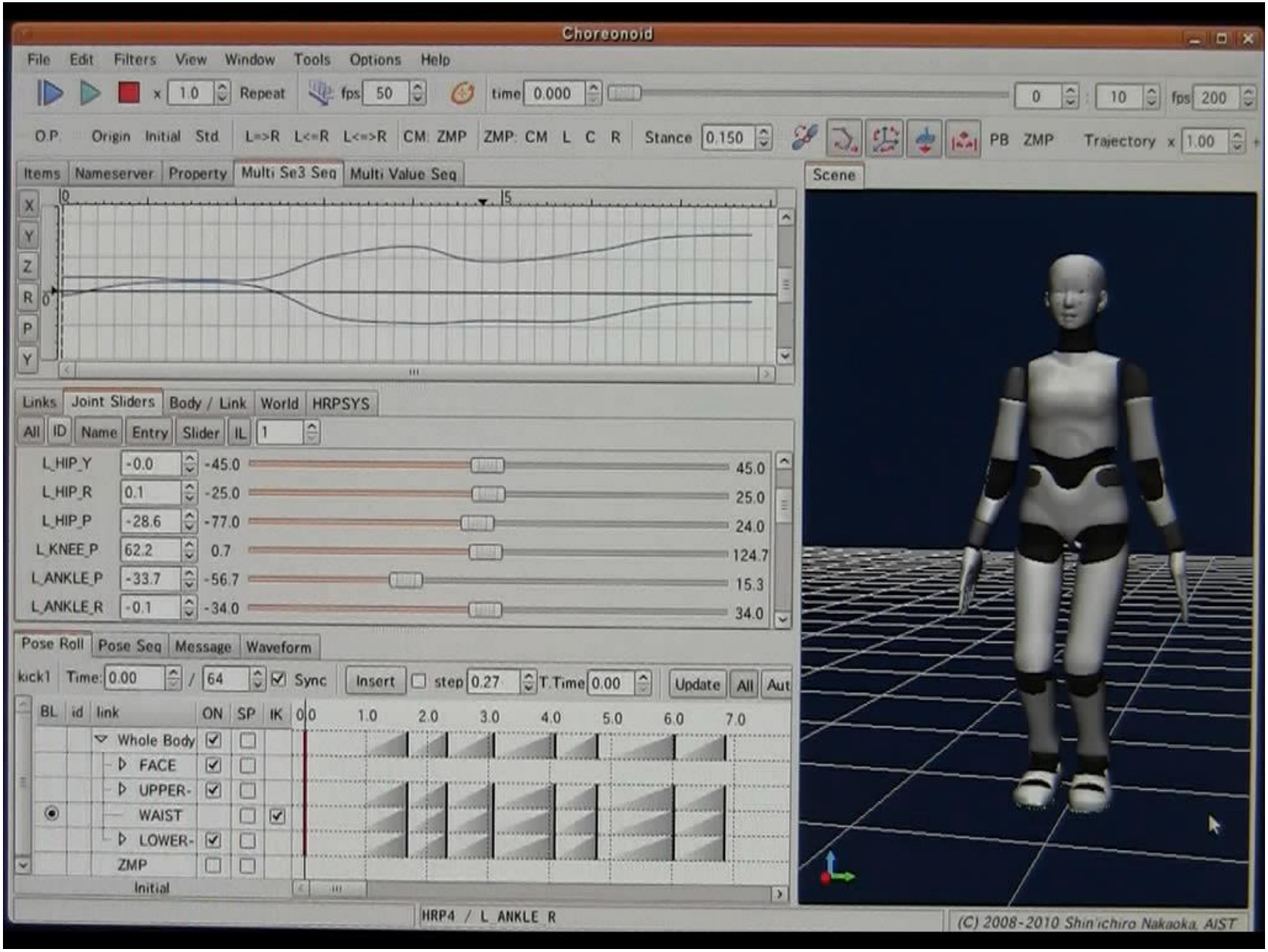
- Choreonoidの概要
  - Choreonoidの開発背景
  - Choreonoidのシステム構成
  - Choreonoidの使い方
- OpenHRIの概要
  - OpenHRIの開発背景
  - OpenHRIで提供されている機能
- ChoreonoidとOpenHRIを用いたロボティクスの研究開発事例
  - HRP-4C
  - GraspPlugin for Choreonoid
  - OpenHRIを使ったコミュニケーションロボット

HRP-4C 動画

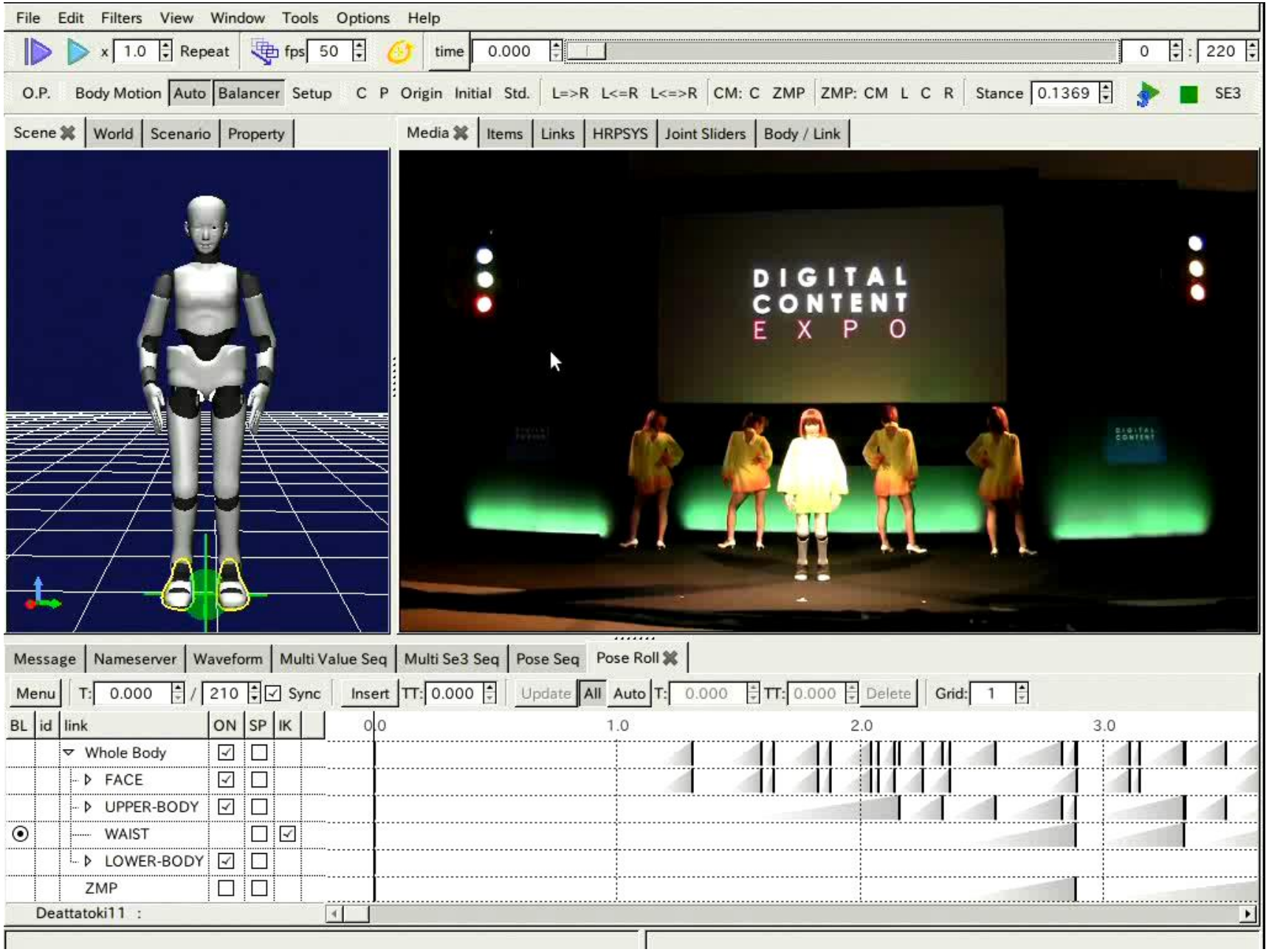
- HRP-4C
  - HRP-4C PressRelease
  - CEATEC
  - 国際ロボット展2011

- HRP-4
  - Press Release

Choreomoidの活用事例（HRP-4C）

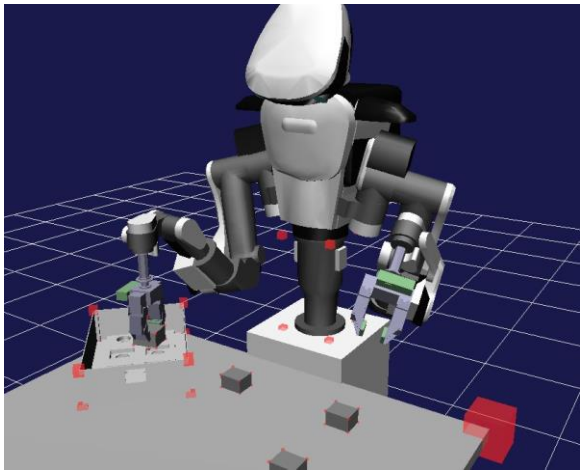


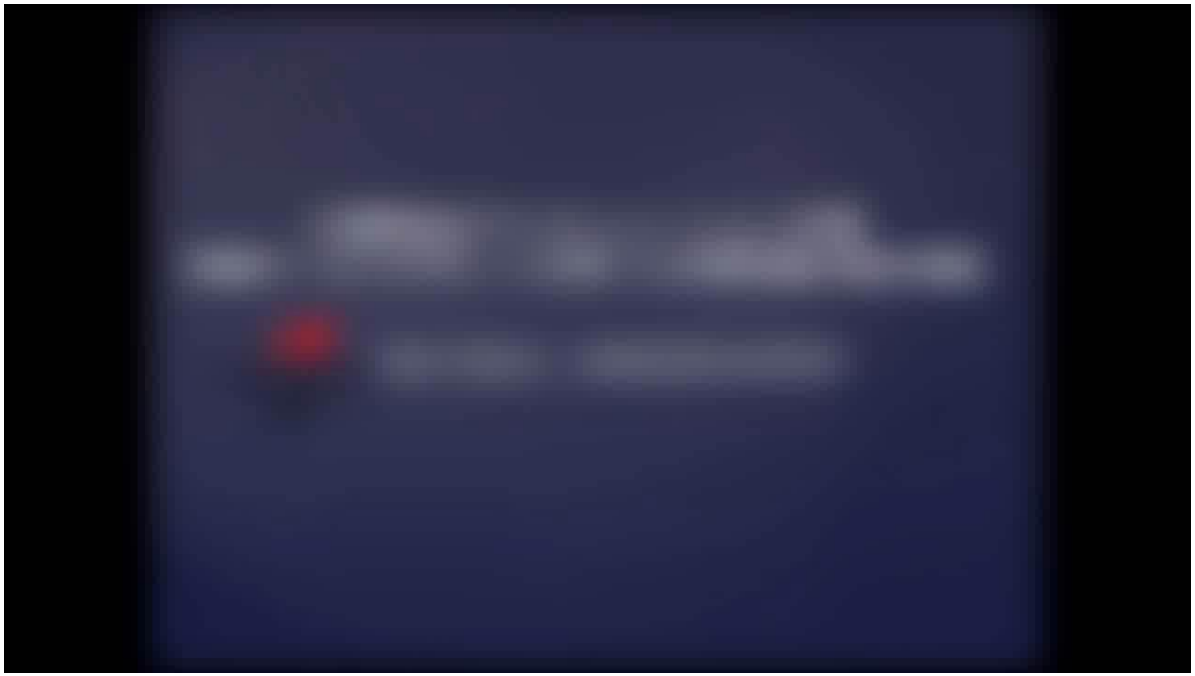
Choreonoidの活用事例(HRP-4C)



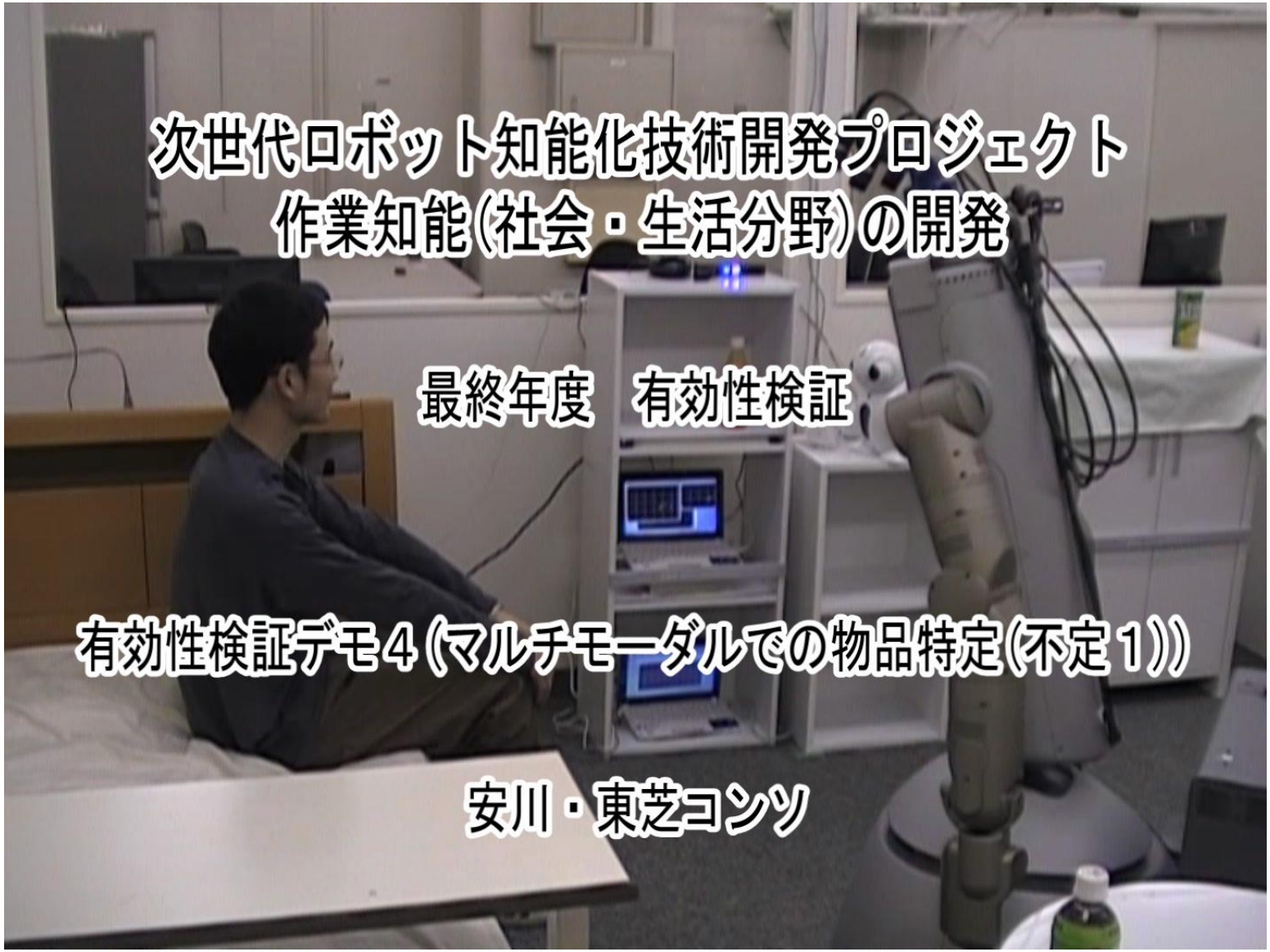
Choreonoidの利用例

- 知能化PJ　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/



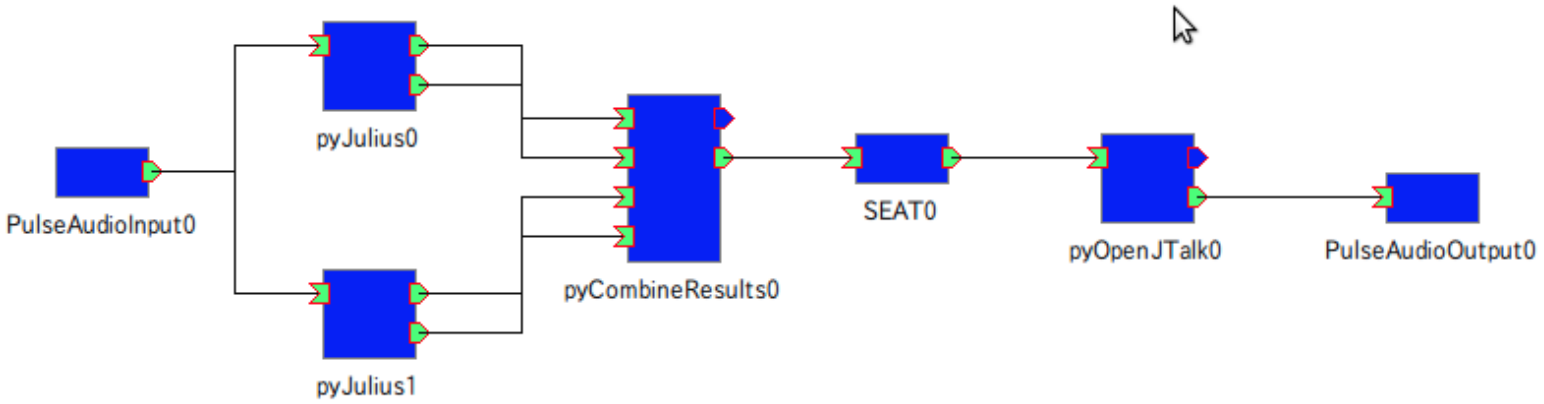


OpenHRIの応用例

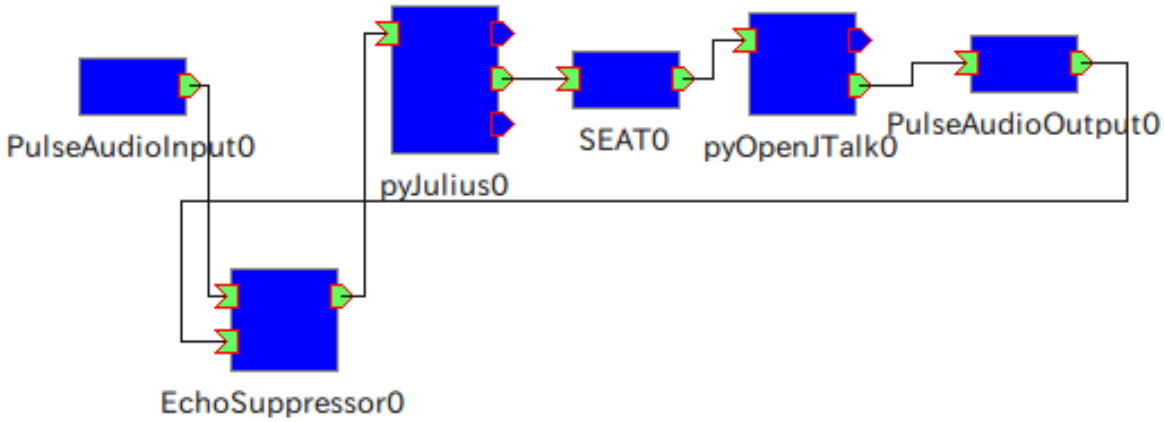


OpenHRIを使ったシステム例

- マルチリンガル対話システム

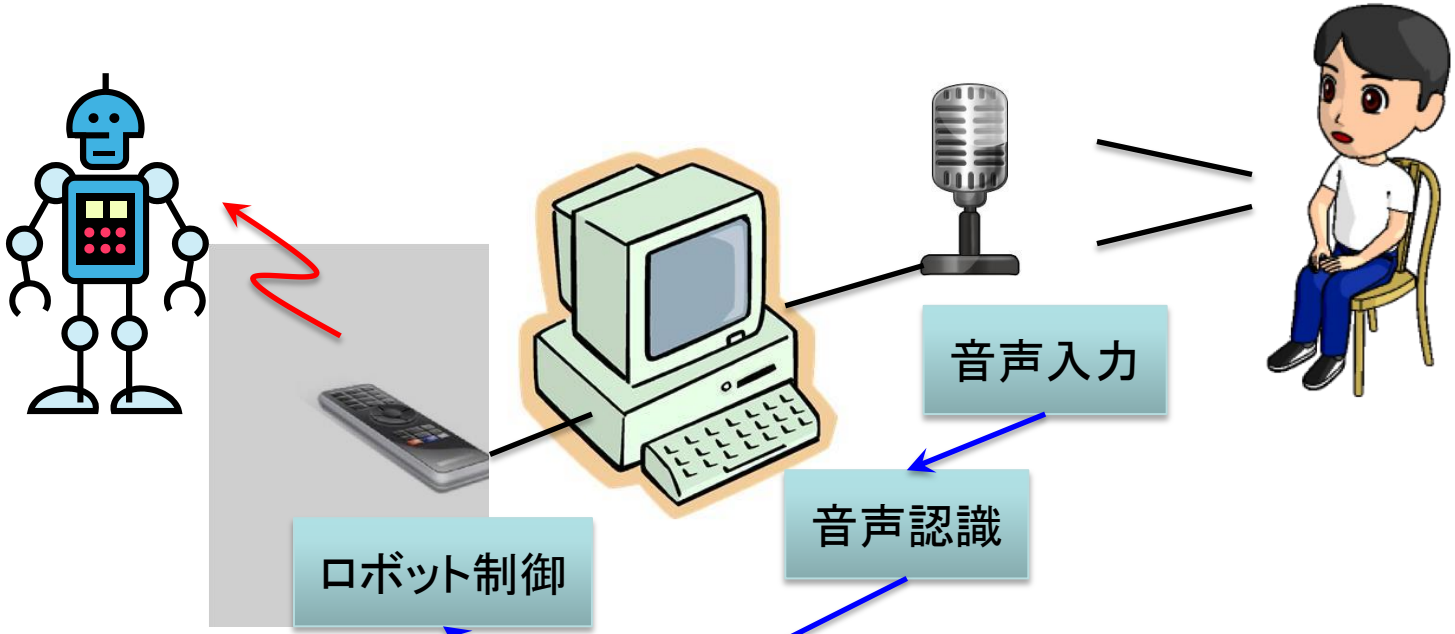


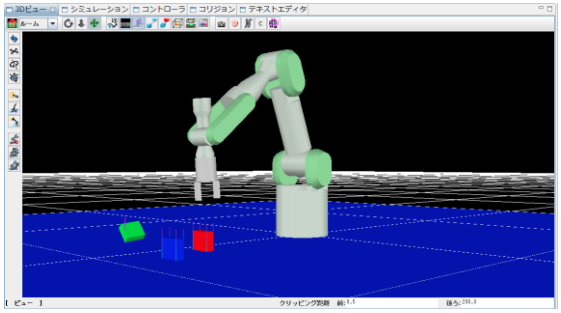
- 音響エコー除去機能を有する対話システム構成



OpenHRIを使ったシステム例

- ロボットに音声インタフェースをつける





i-SOBOTの音声インターフェース例

本来は、、、、

i-SOBOT

PC-OP-RS1

PC

MV100
(Speakerphone)

赤外線
リモコン

赤外線
信号学習

SEAT
(対話制御)

音声
合成

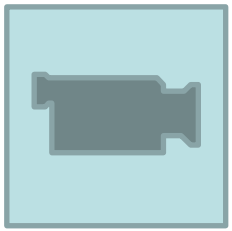
音声
出力

SEATMLに
よるルール
記述

Julius
(音声認識)

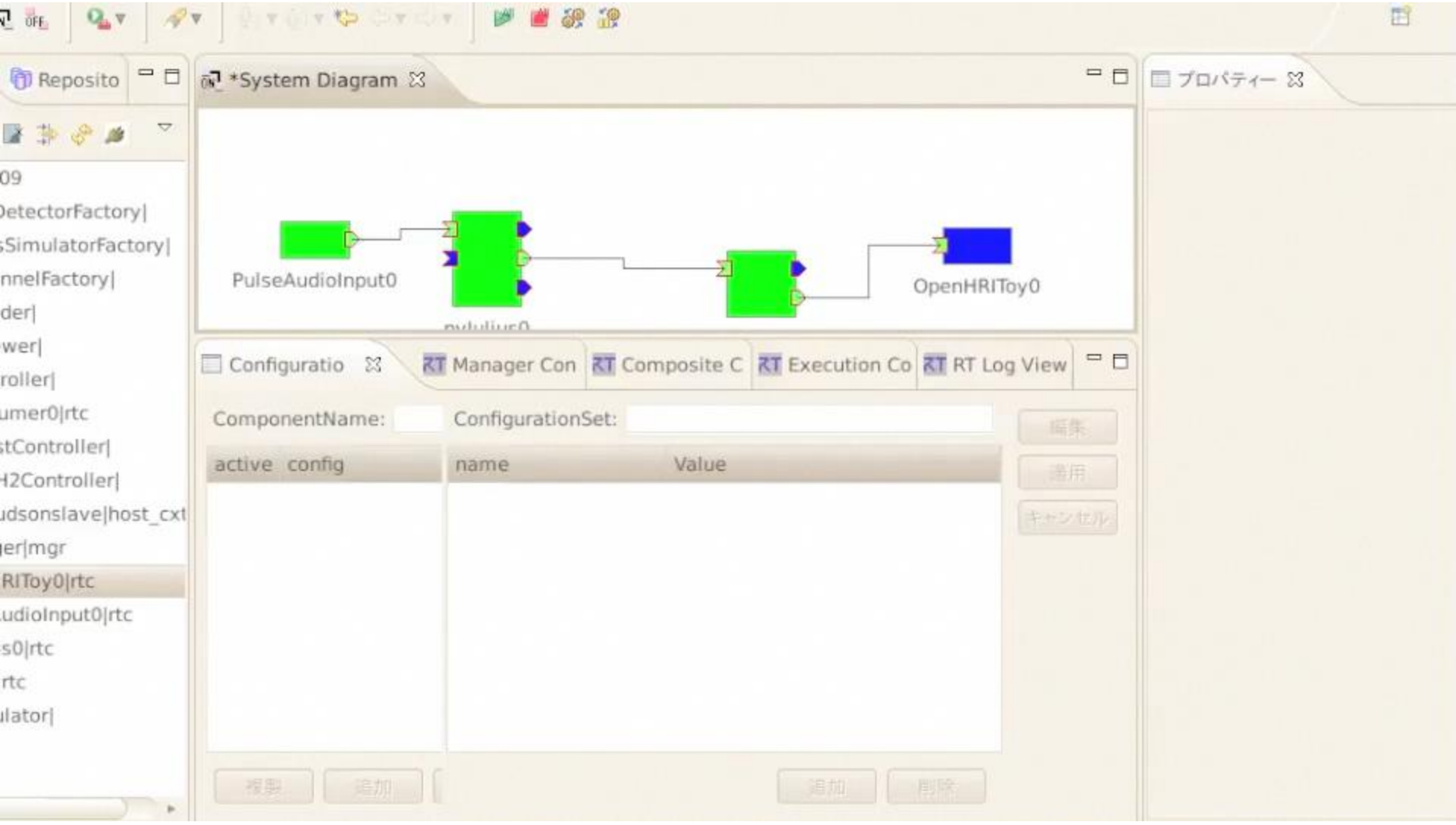
音声
入力

音声認
識文法



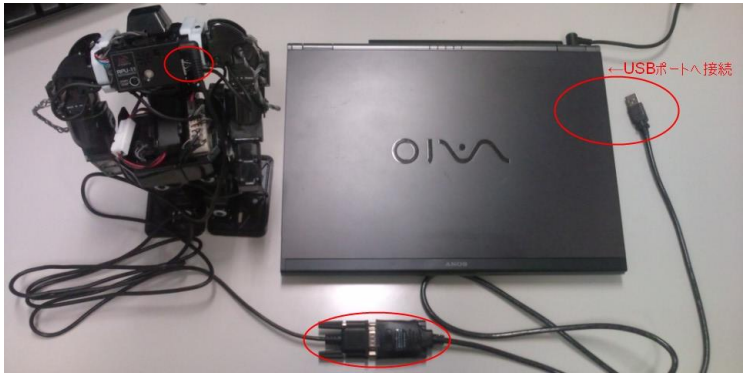
OpenHRIを使ったシステム例

- ロボットシミュレータを使った音声インターフェース



OpenHRI応用例：ロボット音声IF

- 市販のホビーロボットを音声命令で動作させる
- 音声対話モジュールSEATの状態遷移モデル作成の例
- ロボットの状態（姿勢）に応じて、音声コマンドとの対応を変化させる

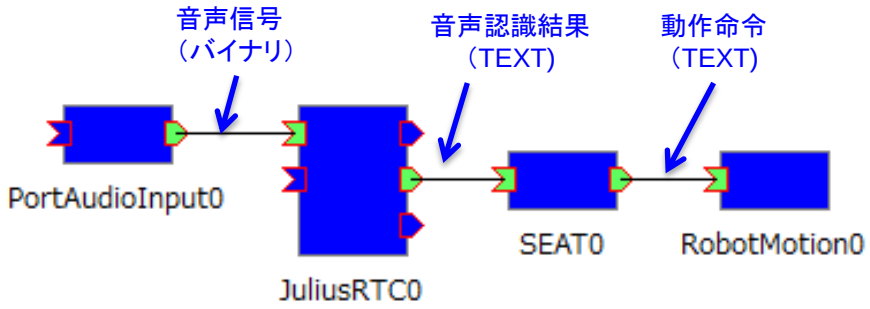


【ハードウェア】

- Windows 7の動作するパソコン
- KINECT
- G-ROBOTS GR-001　または Choreonoid

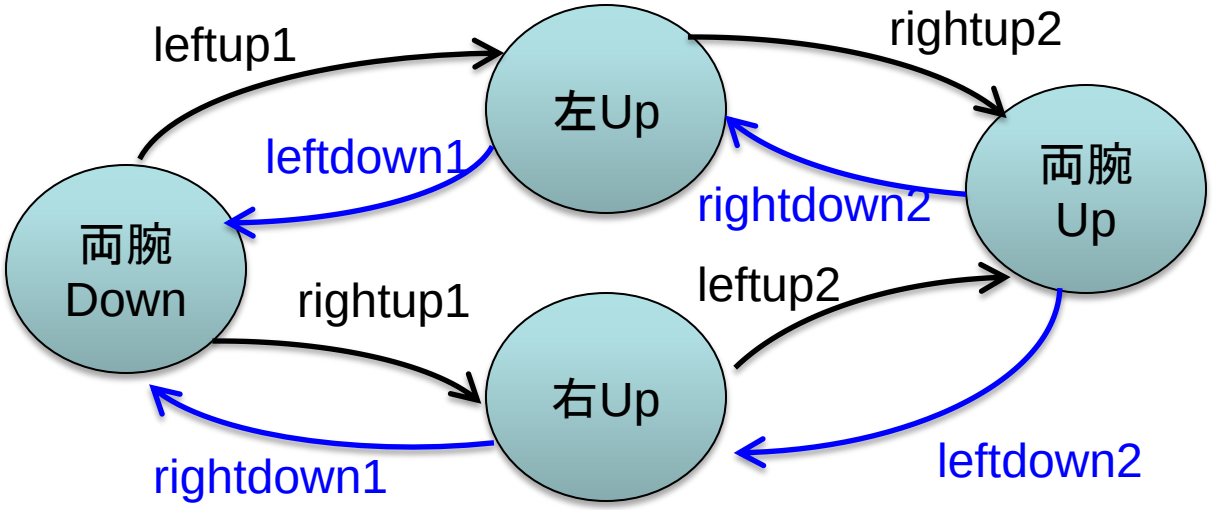
【利用コンポーネント】

- KINECTコンポーネント: 音声データの取得
- Juliusコンポーネント: 日本語音声認識
- SEATコンポーネント : 音声対話制御
- Choreonoid: GR-001シミュレーション



利用するロボットの動作パターン

- Choreonoidで作成した8つの動作
  - leftup1：両腕をおろした状態から左腕を挙げる
  - leftup2:右腕を挙げた状態から左腕を挙げる
  - rightup1：両腕をおろした状態から右腕を挙げる
  - rightup2:左腕を挙げた状態から右腕を挙げる
  - Leftdown1：左腕のみを挙げた状態から左腕をおろす
  - leftdown2:両腕を挙げた状態から左腕を下ろす
  - Rightdown1：右腕のみを挙げた状態から右腕をおろす
  - rightdown2:両腕を挙げた状態から右腕をおろす

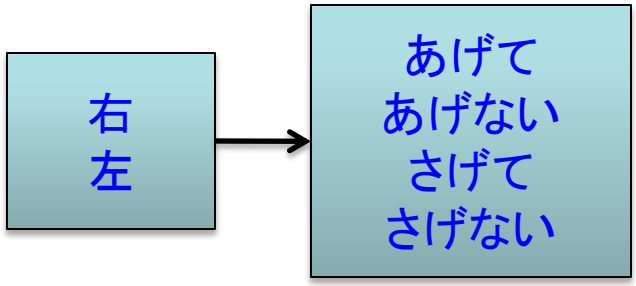


音声認識文法

```
<?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>右</item>
      <item>左</item>
    </one-of>
    <one-of>
      <item>あげて</item>
      <item>あげない</item>
      <item>さげて</item>
      <item>さげない</item>
    </one-of>
  </rule>

</grammar>
```



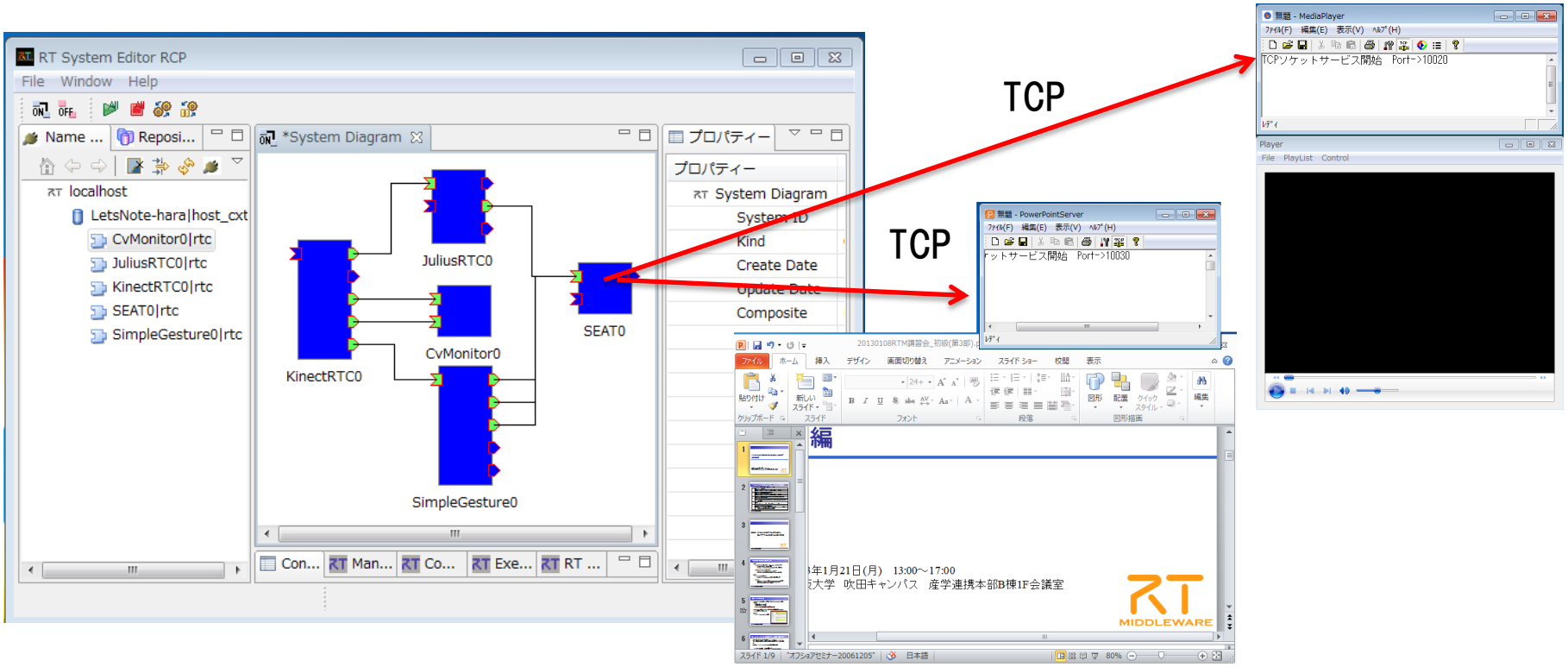
音声対話定義

```
<?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>右 (あげて|さげない)</key>
      <command host="command">rightup1</command>
      <statetransition>right_up</statetransition>
    </rule>
    <rule>
      <key>左 (あげて|さげない)</key>
      <command host="command">leftup1</command>
      <statetransition>left_up</statetransition>
    </rule>
```

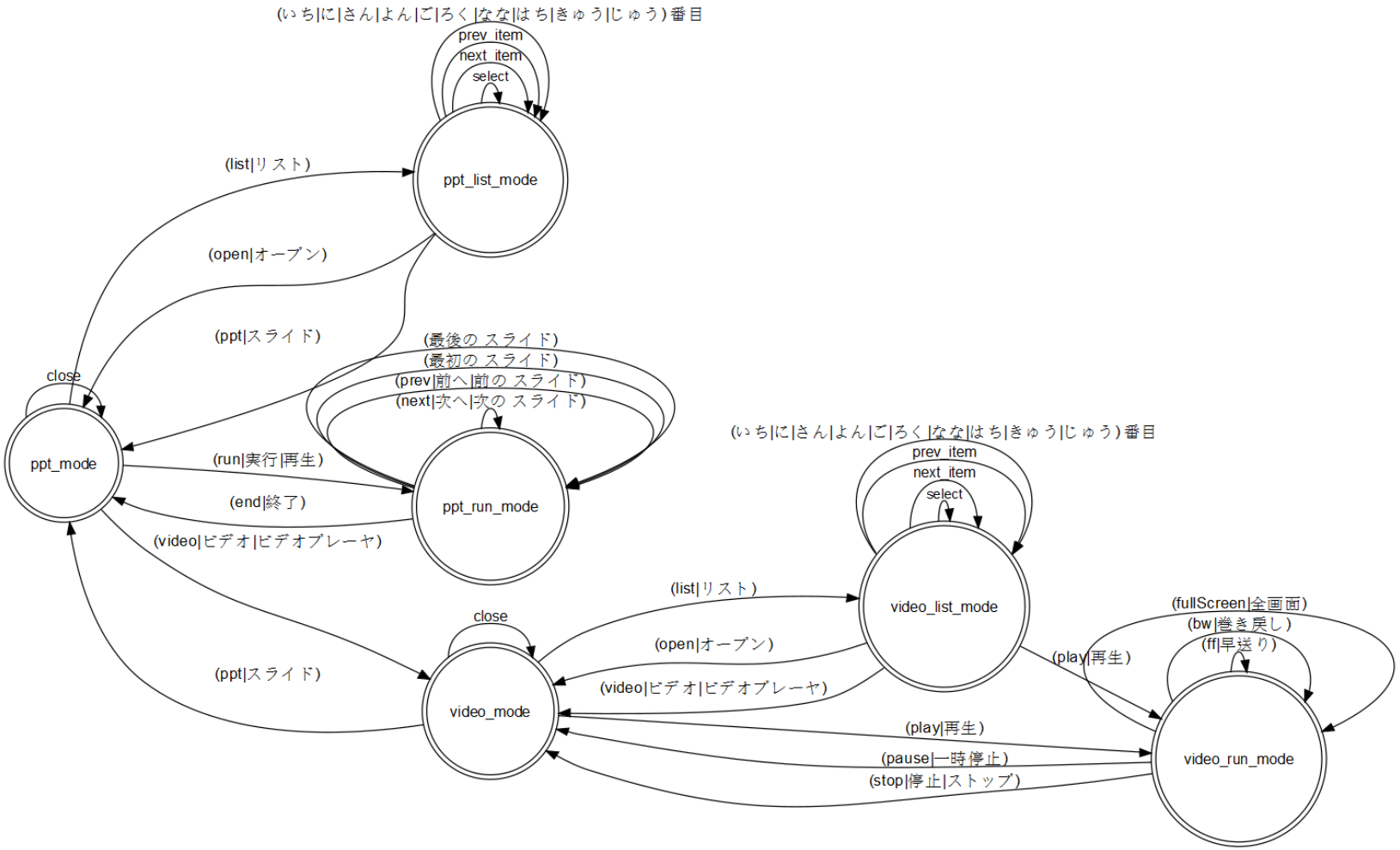
＜中略＞

```
</state>
</seatml>
```

- 音声(ジェスチャ)によるスライド、ビデオの操作
  - 音声認識、KINECTを使ってスライド、ビデオなどプレゼン用ソフトウェアを操作する
  - スライド、ビデオの操作ソフトは、TCP Socketを用いた簡易なもの(過去に開発したソフトウェアの再利用)

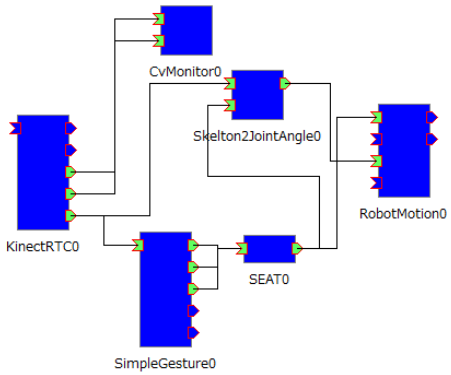


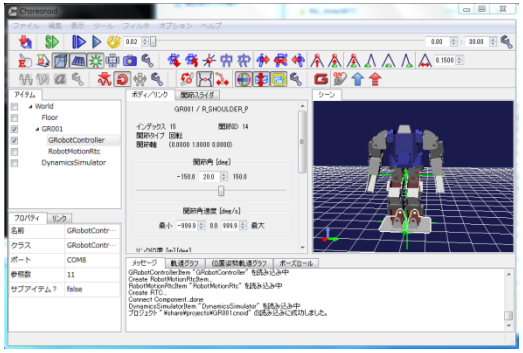
- 音声対話エンジンの状態図

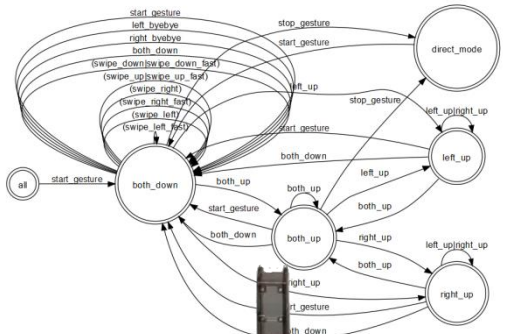


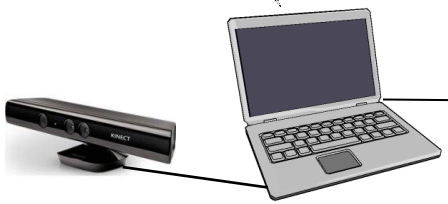
Example System using Kinect Sensor

- Direct control by a human posture
- Command control by a gesture recognition

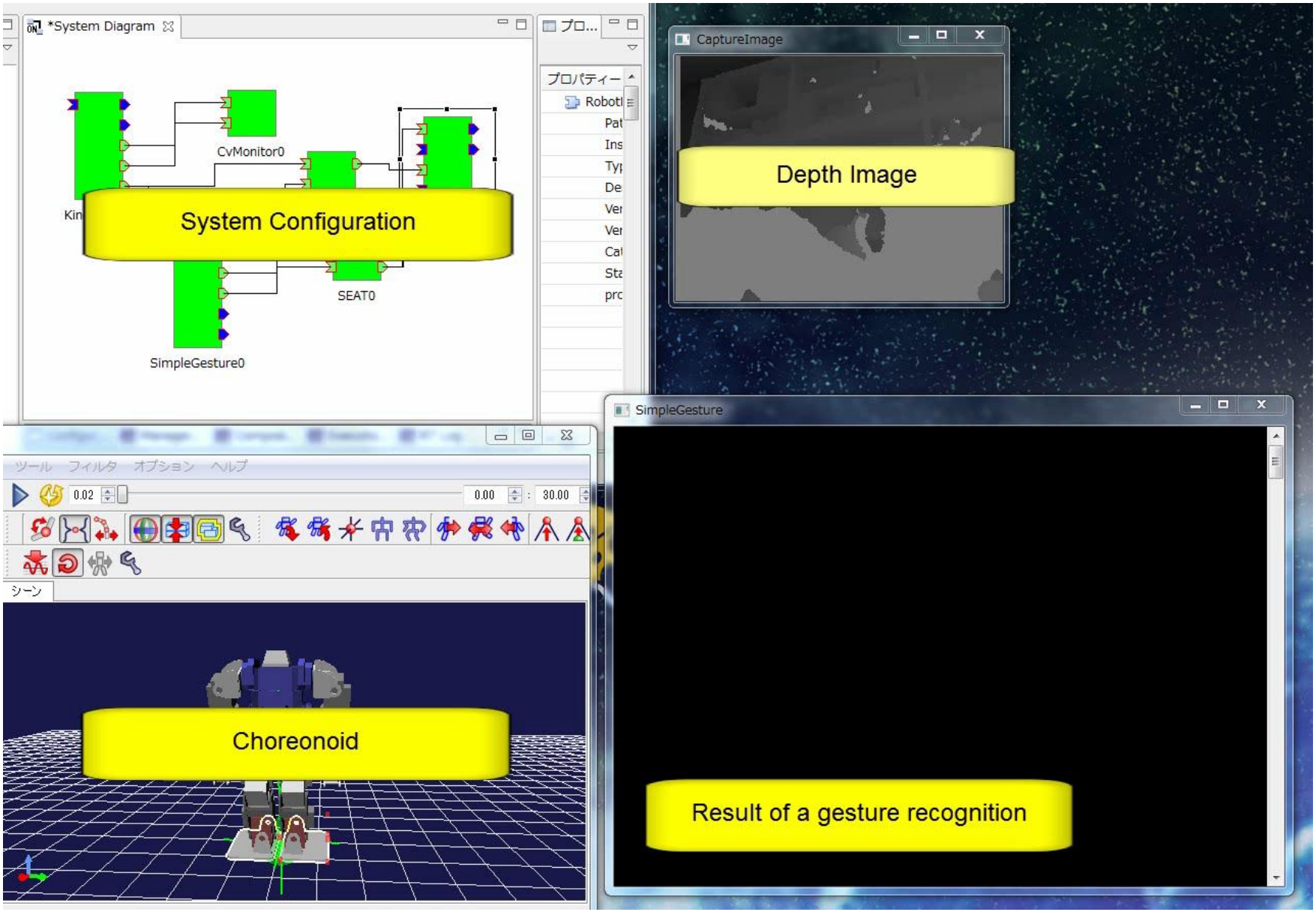








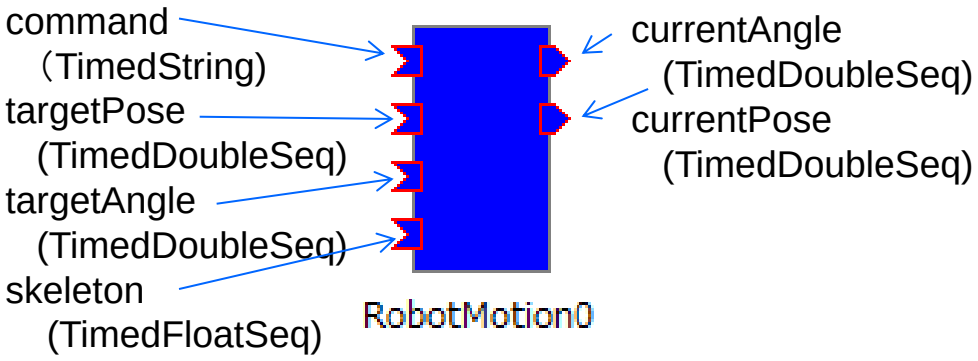
Demonstration



RobotMotion

本コンポーネントは、ChoreonoidのRobotMotionRtcItem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基づいて、
Chorenoid内のロボットモデルの姿勢を変更します。

targetAngleとtargetPoseは、目標関節角を配列にしたデータを入力としますが、単位と対応関節の順番が異なります。
skeletonには、KinectRTCの同名のデータポートと接続し、Kinectで計測されたスケルトン位置データを入力とします。
出力ポートcurrentAngleとcurrentPoseは、現在の関節角を配列にしたデータを出力します。

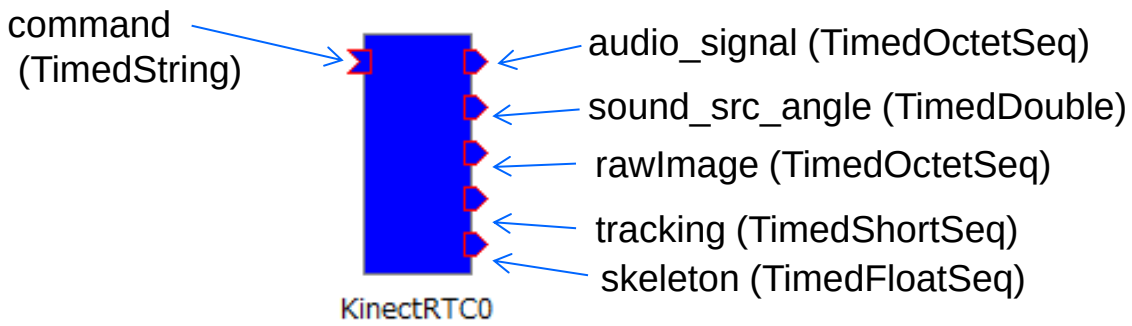


名前	ポート	データ型	説明
command	InPort	TimedString	コマンド(動作パターン名)， 対応ポート:SimpleGesture.command 等
targetPose	InPort	TimedDoubleSeq	G-ROBOTの各関節の目標角度データ(単位:1/10 Degree) データ長:20， 対応ポート:GRobotRTC.qRef
targetAngle	InPort	TimedDoubleSeq	G-ROBOTの各関節の目標角度データ(単位:Radian) データ長:20 or 21， 対応ポート:Skelton2JointAngle.joint_angle
skeleton	InPort	TimedFloatSeq	KINECTで取得した人の姿勢データ データ長:20， 対応ポート:KinectRTC.skeleton
currentAngle	OutPort	TimedDoubleSeq	G-ROBOTの各関節の現在の角度データ(単位:Radian) データ長:20， 対応ポート:Skelton2JointAngle.joint_angle
currentPose	OutPort	TimedDoubleSeq	G-ROBOTの各関節の現在の角度データ(単位:1/10 Degree) データ長:20， 対応ポート:GRobotRTC.q

KinectRTC

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、Kinect for Windows SDKで定義された順番で出力します。trackingは、座標データで x1, y1, x2, y2, … のような配列になっており、skeletonからは、x1, y1, z1, x2, y2, z2, … のようにKINECTで取得した各関節の座標データの配列になります。

また、audio_signalは、OpenHRIの音声入力ポートと互換性をもっており、JuliusRTC等と接続して利用可能です。
rawImageからは、カラー画像または、深度画像を出力します。姿勢データを出力時には、深度画像が出力されます。

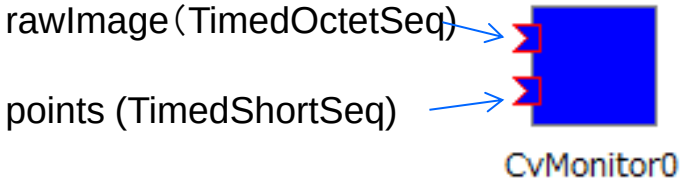


名前	ポート	データ型	説明
command	InPort	TimedString	コマンド入力（未実装）
audio_signal	OutPort	TimedOctetSeq	音声データ (16KHz, 16bits), 対応ポート：JuliusRTC.data
sound_src_angle	OutPort	TimedDouble	音源の方向（Radian）
rawImage	OutPort	TimedOctetSeq	カラー画像または深度画像（Configurationで設定）， 対応ポート：CvMonitor.rawImage
tracking	OutPort	TimedShortSeq	姿勢データ（表示用の座標，　データ長：20）、深度画像取得を設定時のみ 対応ポート：CvMonitor.points
skeleton	OutPort	TimedFloatSeq	姿勢データ（ロボット操作用、データ長:21）、深度画像取得を設定時のみ 対応ポート：Skelton2JointAngle.skeleton,　RobotMotion.skeleton

CvMonitor

本コンポーネントは、KINECTコンポーネント(KinectRTC)からの出力を表示するためのコンポーネントです。rawImageポートへの入力は、RGB24またはRGB32の画像フォーマットのデータのみに対応します。
デフォルトでは、深度画像表示となりますが、コンフィグレーションの設定で、表示画像の大きさとビット数を変更することができます。

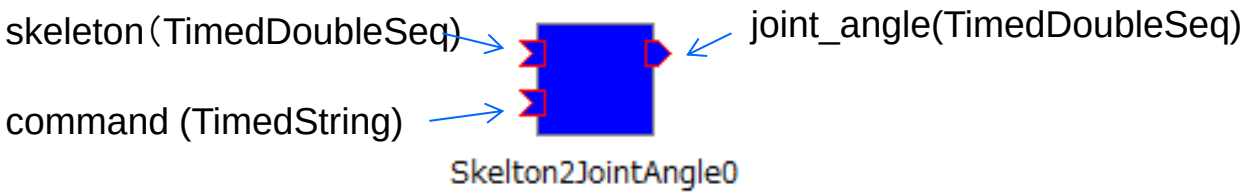
また、pointsポートへの入力は、KinectRTCから出力されるスケルトン位置データのX，Y座標の配列データを前提としています。この入力データに対応した位置に〇が表示されます。



名前	ポート	データ型	説明
rawImage	InPort	TimedOctetSeq	Raw画像データ（表示する画像データ） 対応ポート： KinectRTC.rawImage
points	InPort	TimedShortSeq	KINECTで取得した人の姿勢データ（表示用） 対応ポート：KinectRTC.traking

Skelton2JointAngle

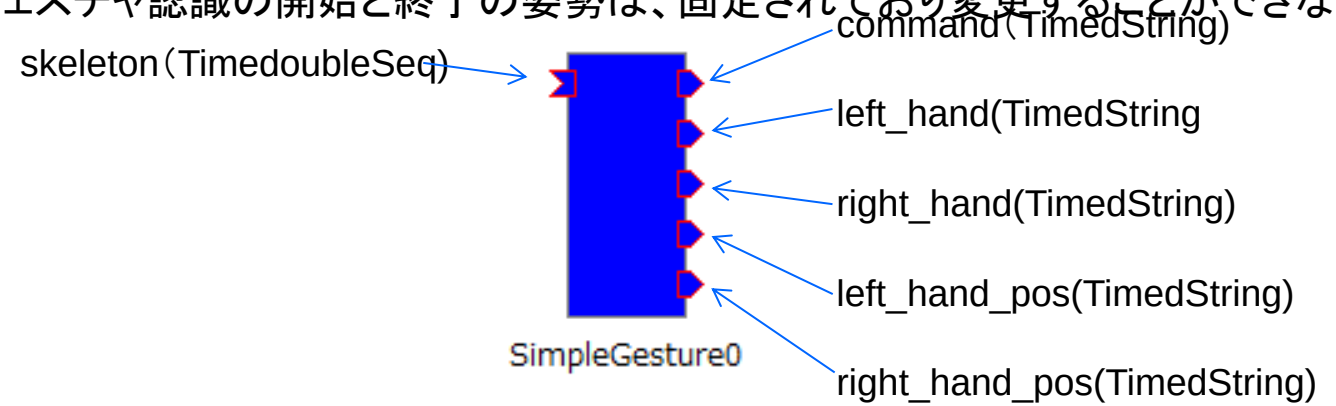
本コンポーネントは、KINECTコンポーネント(KinectRTC)からの出力から各関節の角度へ変換しRobotMotionコンポーネントへ入力するためのデータ変換を行うためのコンポーネントです。
command入力ポートには、startまたはstopのコマンドを受け付け、データ変換のOn/Offを切り替えることができます。
joint_angleから出力されるデータは、RobotMotionコンポーネントのtargetAngle入力ポートへの入力に対応し、各関節への目標角度＋SPINEのY軸方向（高さ方向）の位置データになります。そのためデータ長は、20+1となっています。



名前	ポート	データ型	説明
skeleton	InPort	TimedDoubleSeq	KINECTで取得した人の姿勢データ データ長:20, 対応ポート: KinectRTC.skeleton
command	InPort	TimedString	コマンド入力 対応ポート:SEAT.command
joint_angle	OutPort	TimedDoubleSeq	各関節の角度データ（単位Radian）+SPINEのY軸方向の位置 データ長:21, 対応ポート: RobotMotion.targetAngle

SimpleGesture

本コンポーネントは、KINECTコンポーネント(KinectRTC)からの出力から簡単なジャスチャ認識を行うコンポーネントです。認識結果は、command, left_hand, right_handポートから文字列で出力されます。
この出力文字列は、固定されていますので、ロボットに対するコマンドを変更するには、SEATコンポーネントなどを使って、変換すると便利です。
また、ジェスチャ認識の開始と終了の姿勢は、固定されており変更することができなくなっています。

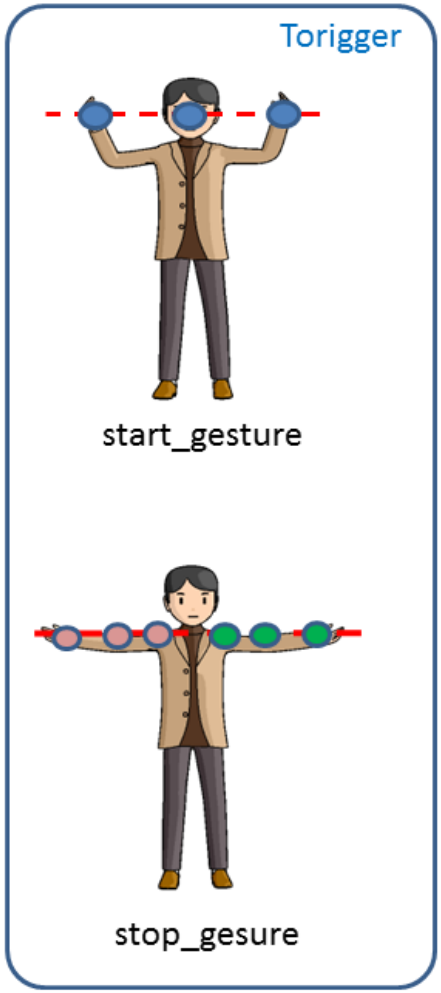


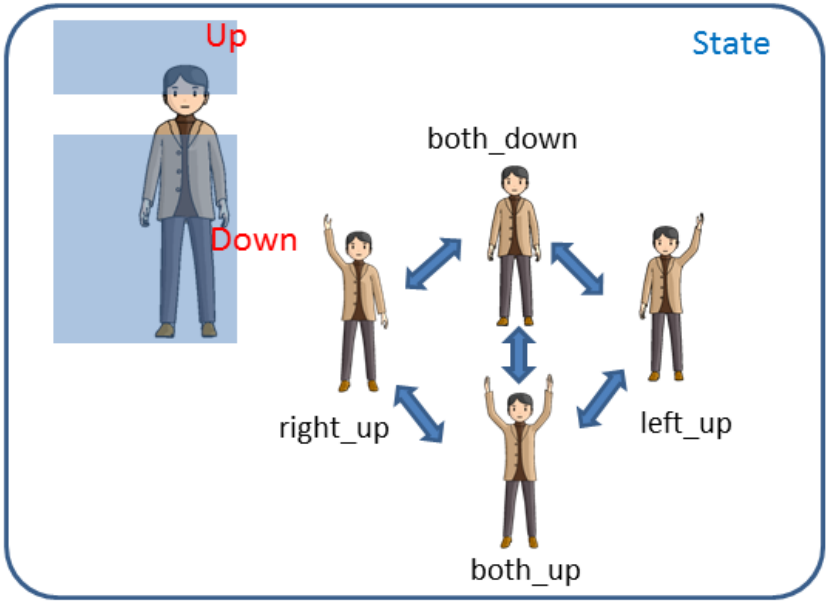
名前	ポート	データ型	説明
skeleton	InPort	TimedOctetSeq	KINECTで取得した人の姿勢データ, データ長:20 対応ポート：KinectRTC.skeleton
command	OutPort	TimedString	ジェスチャ認識結果 対応ポート：SEAT.gestureなど
left_hand	OutPort	TimedString	左手のジェスチャ認識領域での左手の動作認識結果 対応ポート：SEAT.gestureなど
right_hand	OutPort	TimedString	右手のジェスチャ認識領域での右手の動作認識結果 対応ポート：SEAT.gestureなど
left_hand_pos	OutPort	TimedDouble	左手がStay, Movingの状態時の左肩からの距離（右方向が正）
right_hand_pos	OutPort	TimedDouble	右手がStay, Movingの状態時の右肩からの距離（右方向が正）

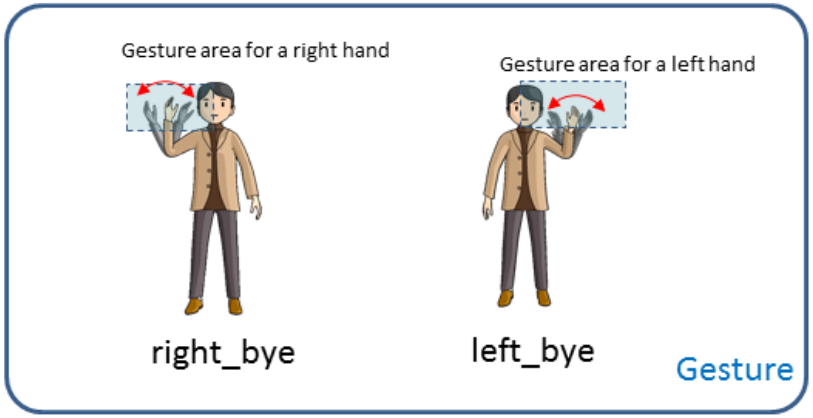
SimpleGesture.command

ジェスチャ認識の開始、終了姿勢とcommandポートから出力される認識結果。左右の手の状態認識は、ほぼ顔の位置の横側。

DataOutPort: Command

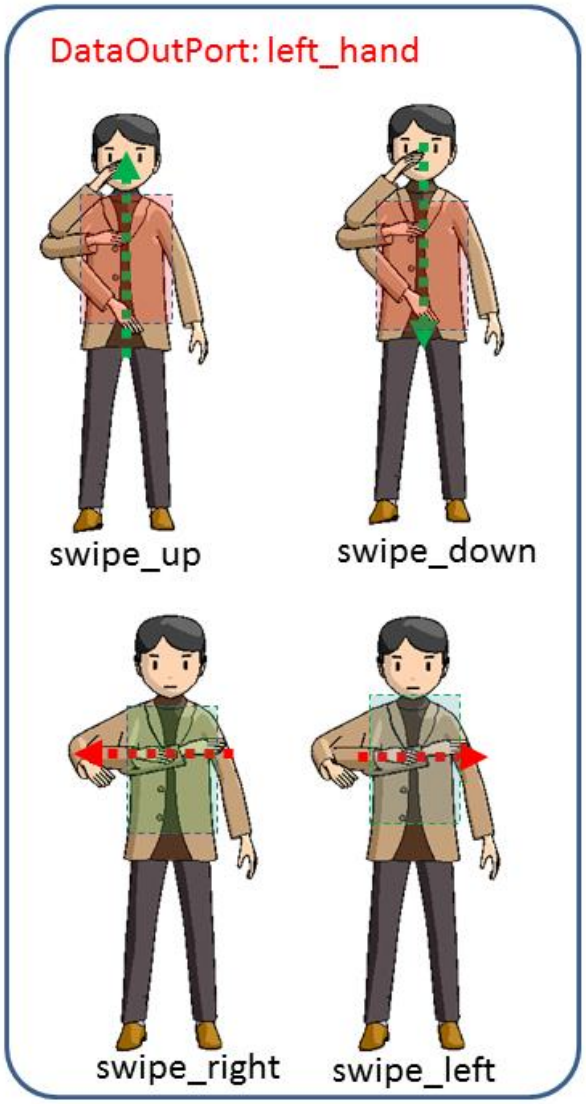


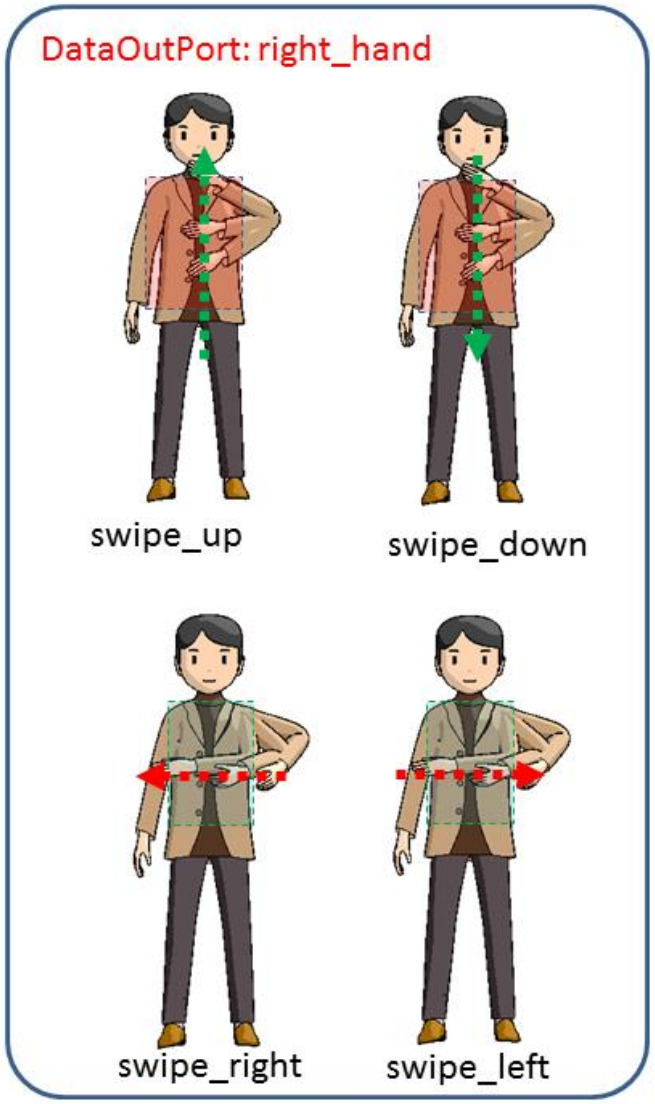




SimpleGesture.left_hand,right_hand

left_hand, right_handポートから出力される認識結果。ジェスチャ認識領域は、赤枠（または青枠）内





おわりに

- Choreonoidとシステム事例の紹介
  - 動力学シミュレーションを行いながら多関節型ロボットの動作を簡単に記述
  - 直感的で簡易なインターフェースを提供

- OpenHRIとシステム事例の紹介
  - OSS版コミュニケーション知能RTC
  - 対話システムを簡単に構成するためのコンポーネント群
  - 音声認識と対話制御の簡素化