RTコンポーネント作成応用編

サービスポートの作り方

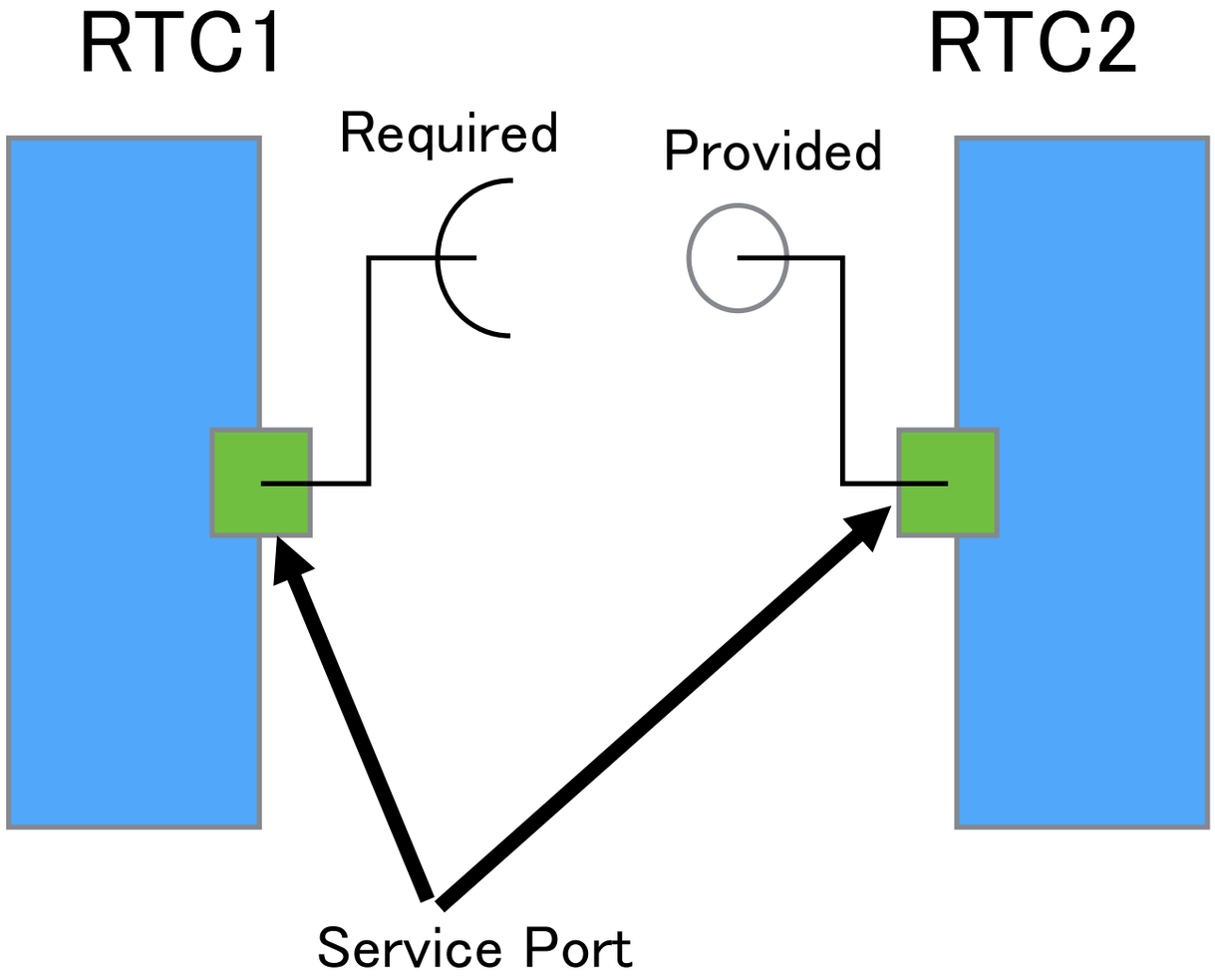
株式会社SSR × 早大基幹理工表現工学科尾形研究室
菅　佑樹

サービスポートの概念

- OpenRTM-aistの機能
- RTCに関数呼び出しのインターフェースを付加
  - 引数でデータを渡す
  - 返り値でデータを受け取る
- データポートと比べて
  - データを要求することができる
  - データの送受信の結果が得られる（成否など）
  - データの送受信に関わらない処理を実現できる
    - reset, initializeなど
  - 若干実装が面倒

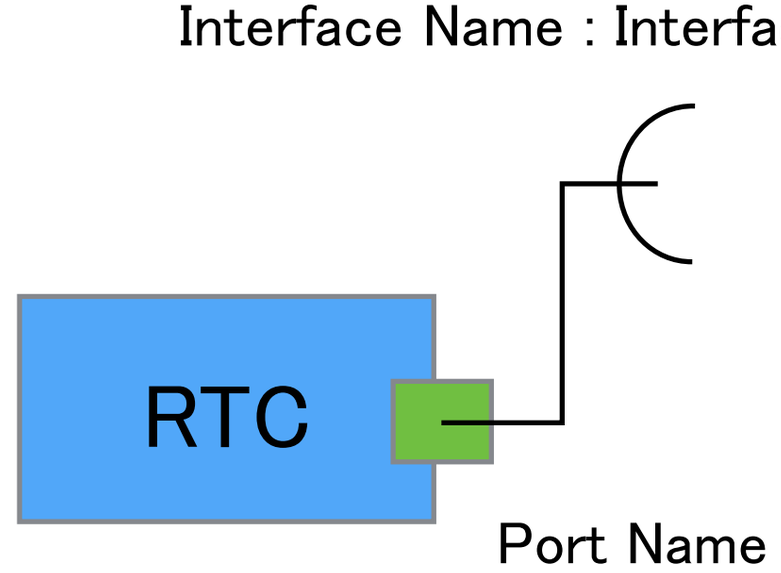
サービスポートの概念

- サービスポートは「インターフェース」を持つ
- インターフェースには極性 (Polarity) がある
  - 提供 (Provided)
    - リソースを提供する．関数が呼ばれる側.
  - 要求 (Required)
    - リソースを利用する．関数を呼ぶ側.
- 一つのサービスポートが複数の提供・要求インターフェースを持つことができる
- インターフェースのタイプが合えば接続でき，利用できる.



サービスポート作成の流れ

- サービスポートを追加
  - 設定項目
    - サービスポート名
  - インターフェースを追加
    - 設定項目
      - インターフェース名*
      - インスタンス名
      - 極性 (Polarity)
      - インターフェース型
- 注意！
  - インターフェース名が違うと繋げるのが面倒になる！
- インターフェース型を定義するファイルを作成 (IDLファイル)



Interface Definition Language

- サービスポートの関数やデータ型を宣言するためのファイル形式 (.idlファイル)
  - CORBAやDDSなどの分散処理実現のために使われる
  - Javaに似た形式
  - オブジェクト指向
- IDLファイルをコンパイルすると，C++やJava, Pythonのプログラムのスケルトンが生成される
  - IDLは中間言語
  - C++やJavaなどの実装言語に変換すると，各言語ごとに若干利用方法が変わるが，関数名などは変わらない
  - 今回はC++版を少しだけ解説

IDLの書き方

- moduleで名前空間を定義
  - 後述のインターフェース名は同じ名前になりがちなので名前空間で分ける
- interfaceでインターフェース名を定義
  - 機能をグループ化
- interface内で関数を定義
- 返り値，引数リスト，Javaとほぼ同じ構文
- 引数にin/outを設定
  - in ・・・インターフェース提供側に引数を渡す (値渡し)
  - out ・・・インターフェース提供側が引数にデータを設定する (参照渡し)
  - inout ・・・インターフェース提供側に引数データを渡し，提供側がさらにそれを変更する

```
// 行コメント

// moduleはC++のネームスペースに似た概念
module my_module {


    // interfaceはJavaのインターフェース，
    // C++の仮想クラスに似ている．
    // メンバ関数を定義する
    interface MyInterface {


        // 返り値でデータを受け取るタイプの関数
        long getLongData();

        // 引数でデータを与えるタイプの関数
        void setLongData(in long data);

        // 引数に参照渡しでデータを
        // 受け取ることもできる
        void getDataByArg(out long data);

    };
};
```

IDLの書き方

- structで構造体を定義できる
  - 異なるタイプのデータの集まり
- 引数にも，返り値にも使える

- structでデータ型を定義できれば，データポートの独自データ型作成ができる
  - 参考: http://ysuga.net/?p=213

```
module my_module {

    // 構造体を宣言することでプログラムを構造化できる
    struct MyData {
        long data1;
        long data2;
        double data3;
    };

    interface MyInterface {

        // 返り値でデータを受け取るタイプの関数
        MyData getMyData();

        // 引数でデータを与えるタイプの関数
        void setMyData(in MyData data);

        // 引数に参照渡しでデータを受け取ることもできる
        void getMyDataByArg(out MyData data);

    };
};
```

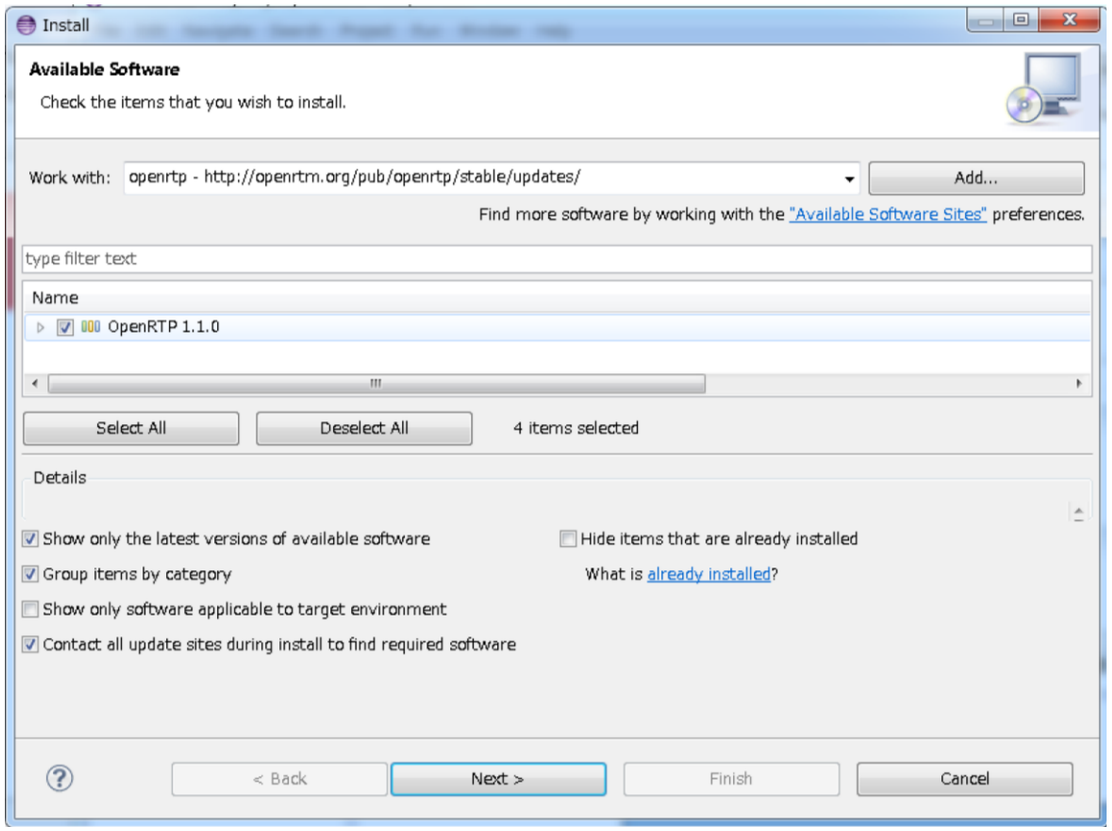
IDLの書き方

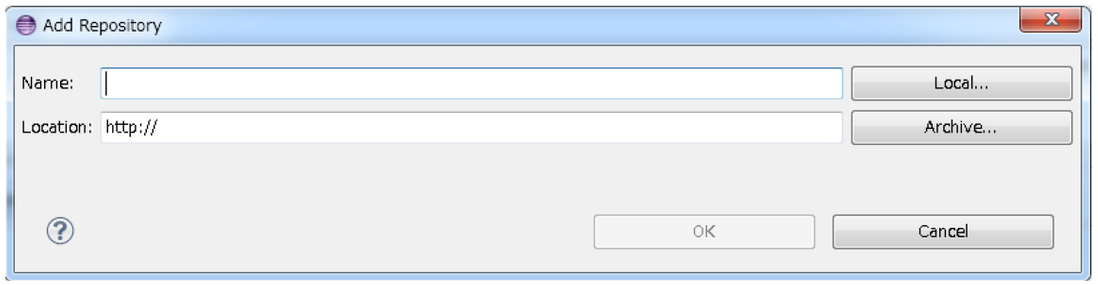
- typedefでデータ型の名称変更
- enumで定数を作る
- sequenceで可変長のデータを作成できる

```
// typedef を使ってラベルを変更できる.
typedef long ERROR_CODE;

// enumを使って数にラベルを付けられる
enum MY_STATE {
    STATE_RUNNING,
    STATE_HALT,
    STATE_ERROR
};

enum RETURN_CODE {
    RETURN_OK,
    RETURN_ERROR
};

struct MyData {

    ERROR_CODE errorCode;

    My_STATE state;

    sequence<double> data;
    // sequenceを使うと可変長の配列を使う
};

interface MyInterface {

    // 引数でデータを与えるタイプの関数
    // 返り値で処理の成否を返すことができる
    RETURN_CODE setMyData(in MyData data);

    // 引数に参照渡しでデータを受け取ることもできる
    // 返り値で処理の成否を返すことができる
    RETURN_CODE getMyDataByArg(out MyData data);
};
```

RTC Builderの使い方

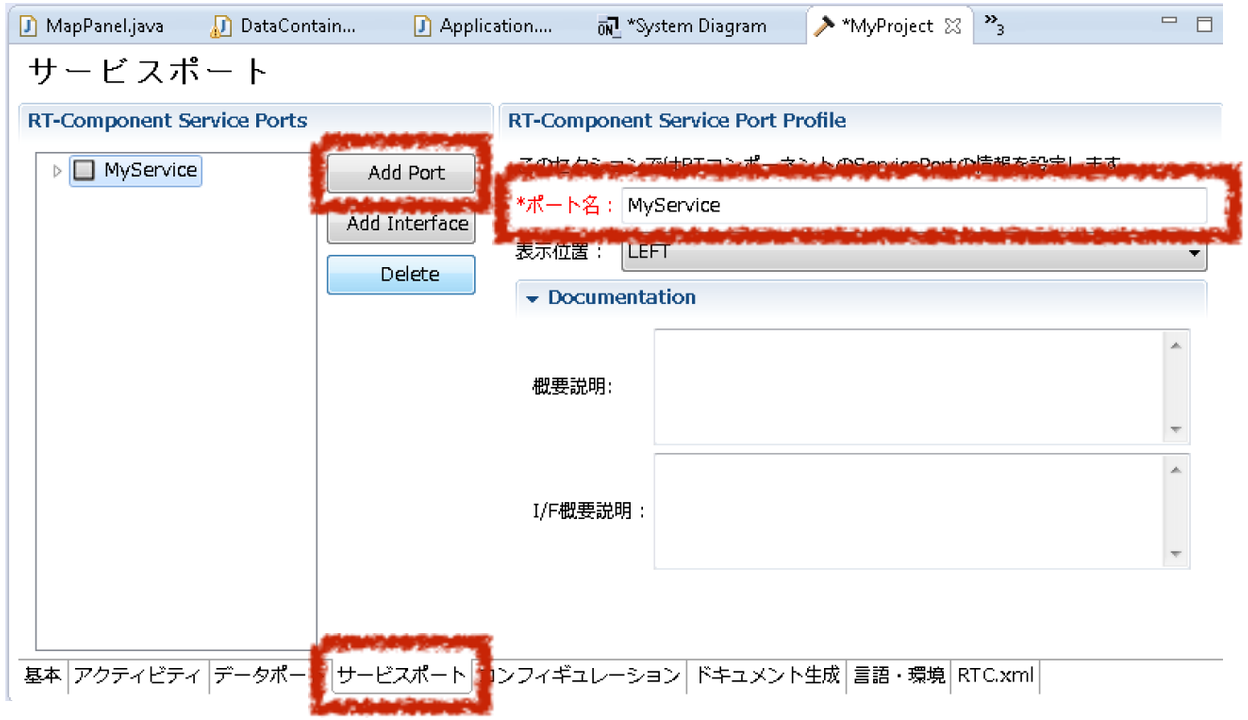
- Eclipseのプラグインのアップデートが必要
- メニュー ＞ Help ＞ 「Install New Software」
  - 起動したダイアログの「Add」を押す
  - Locationに「 http://openrtm.org/pub/openrtp/stable/updates/ 」と入力
- OpenRTP 1.1.0 を選択して「Next」からインストールする



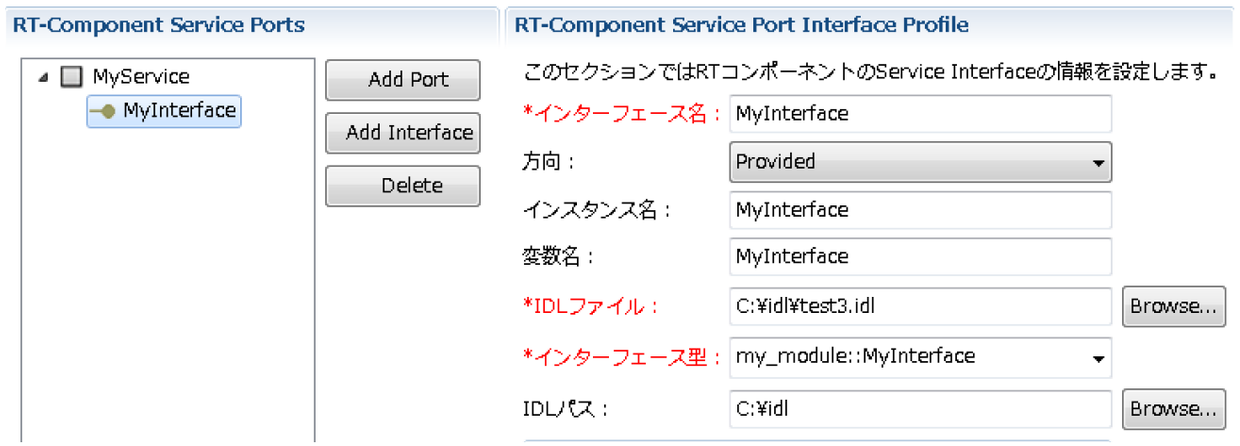


RTC Builder

- 「サービスポート」タブ選択
- 「Add Port」でポート追加
- 「ポート名」を設定



- 「Add Interface」でインターフェース追加
- 「インターフェース名」「方向」「インスタンス名」「変数名」を設定
- 「IDLファイル」の「Browse」ボタンでIDLを読み込む
- 「インターフェース型」を選択
- 「IDLパス」は「IDLファイルがあったフォルダを選択」



ビルド方法

- CMake
  - RTCBuilderが古いと失敗する
- Visual Studio
  - この中でomniidlというコマンドでIDLをコンパイルするが，Python版がインストールされていると問題が起こることがある
    - PATHの設定によっては，Python版のomniidlが優先して見つかるため，C++にコンパイルできない
    - → PATHの最初にC++版のomniidl.exeのPATHを設定しなおす
  - IDLはビルドできてる (** Not Foundのようなエラーは無い) のにコンパイルできない
    - サービスポートを簡易に実行するためのヘルパークラスが生成されるが，独自のIDLパーサーを使っているため問題が多い
      - → 最初は構造体などをあまり使わずに
      - → openrtmのメーリングリストに通報
      - → IDLコンパイルでidlファイル名.hhというファイルが生成される．その中のインターフェースをヘルパークラスがオーバーライドしているので，返り値の型などを手動で直すとコンパイルできることがある

コーディング方法例 (提供側)

```
module my_module {

	// typedef を使ってラベルを変更できる.
	typedef long ERROR_CODE;

	// enumを使って数にラベルを付けられる
	enum MY_STATE {
		STATE_RUNNING,
		STATE_HALT,
		STATE_ERROR
	};

	enum RETURN_CODE {
		RETURN_OK,
		RETURN_ERROR
	};

	struct MyData {

		ERROR_CODE errorCode;

		My_STATE state;

		sequence<double> data;
		// sequenceを使うと可変長の配列を使う
	};

	interface MyInterface {

		// 引数でデータを与えるタイプの関数
		// 返り値で処理の成否を返すことができる
		RETURN_CODE setMyData(in MyData data);

		// 引数に参照渡しでデータを受け取ることもできる
		// 返り値で処理の成否を返すことができる
		RETURN_CODE getMyDataByArg(out MyData data);
	};
};
```

```
my_module::RETURN_CODE MyInterfaceSVC_impl::setMyData(const my_module::MyData& data)
{
	my_module::RETURN_CODE result;

	if (data.errorCode == 0) {
		result = my_module::RETURN_OK;
	} else {
		result = my_module::RETURN_ERROR;
	}

	for(int i = 0;i < data.data.length();i++) {
		std::cout << "Data " << i << " is " << data.data[i] << std::endl;
	}

	return result;
}

my_module::RETURN_CODE MyInterfaceSVC_impl::getMyDataByArg(my_module::MyData_out data)
{
	my_module::RETURN_CODE result = my_module::RETURN_OK;
	my_module::MyData_var myData = new my_module::MyData;

	myData->state = my_module::STATE_RUNNING;
	myData->errorCode = 0;
	myData->data.length(4);
	myData->data[0] = 0.1;
	myData->data[1] = 0.3;
	myData->data[2] = 0.5;
	myData->data[3] = 0.7;

	data = myData._retn();

	return result;
}
```

シンプルな関数呼び出しになる

out型引数は_var型を使う

菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

コーディング方法例 (要求側)

```
module my_module {

	// typedef を使ってラベルを変更できる.
	typedef long ERROR_CODE;

	// enumを使って数にラベルを付けられる
	enum MY_STATE {
		STATE_RUNNING,
		STATE_HALT,
		STATE_ERROR
	};

	enum RETURN_CODE {
		RETURN_OK,
		RETURN_ERROR
	};

	struct MyData {

		ERROR_CODE errorCode;

		My_STATE state;

		sequence<double> data;
		// sequenceを使うと可変長の配列を使う
	};

	interface MyInterface {

		// 引数でデータを与えるタイプの関数
		// 返り値で処理の成否を返すことができる
		RETURN_CODE setMyData(in MyData data);

		// 引数に参照渡しでデータを受け取ることもできる
		// 返り値で処理の成否を返すことができる
		RETURN_CODE getMyDataByArg(out MyData data);
	};
};
```

```
my_module::MyData data;
data.errorCode = 0;
data.state = my_module::STATE_HALT;
data.data.length(3);
data.data[0] = 1.2;
data.data[1] = 1.4;
data.data[2] = 1.8;

if(m_variableName->setMyData(data) != my_module::RETURN_OK) {
    std::cout << "[YourModule] setMyData failed." << std::endl;
}
```

```
my_module::MyData_var yourData = new my_module::MyData();
if(m_variableName->getMyDataByArg(yourData) != my_module::RETURN_OK) {
    std::cout << "[YourModule] getMyData failed." << std::endl;
}

std::cout << "[YourModule] getMyData succeeded." << std::endl;
std::cout << " errorCode : " << yourData->errorCode << std::endl;
std::cout << " state     : " << yourData->state << std::endl;
std::cout << " data      : [";
for(int i = 0;i < yourData->data.length();i++) {
    std::cout << yourData->data[i] << ", ";
}
std::cout << "]" << std::endl;
```

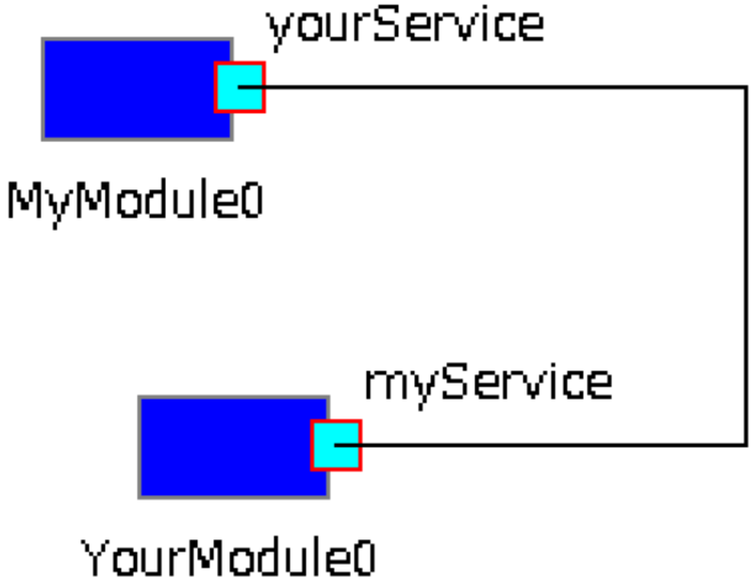
シンプルな関数呼び出しになる

out型引数は_var型を使う

菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

サービスポートの接続

- インターフェース名が同じであれば，問題なく接続できる．
- インターフェース名が違う場合は，「接続できるインターフェースがない」と言われるが，詳細ダイアログで接続するインターフェースを選択すると接続できるようになる．



まとめ

- サービスポートについて紹介
- サービスポートの作成方法について紹介
  - IDLについて紹介
  - C++版のサービスポートの実装方法について紹介

RTM対応 移動ロボット ナビゲーション フレームワーク

株式会社SSR × 早大基幹理工表現工学科尾形研究室
菅　佑樹

RTM対応　移動ロボット ナビゲーションフレームワーク

- 台車型移動ロボットのナビゲーションが目的
  - すべてオープンソース
- マップ作成
  - SLAMのみ
- 地図を用いたナビゲーション
  - レーザーレンジセンサーを用いた位置推定
  - 軌道計画
  - 軌道追従
- 機能ごとにモジュール化しており，インターフェースを共通化して，入替えが可能
  - 例：独自の移動ロボットを適用
  - 例：独自のプランニング・パスフォロワーを適用
- 基本操作が可能なGUIを準備

菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

RTM対応　移動ロボット ナビゲーションフレームワーク MRPT版

- 環境
  - C++版
    - OpenRTM-aist 1.1.0 C++ VC2010版
    - MRPT 1.0.2 for Visual Studio 2010版
    - Visual C++ 2010
    - その他，OpenRTM-aist開発に必要なツール
  - Java版
    - OpenRTM-aist 1.1.0 RC1

フレームワークの特徴

- 上位のインターフェースはすべてサービスポート
- 処理の成功・失敗を返り値で受け取ることができる
- サービスロボットの上位処理はデータフロー的ではなく，イベント的
- MobileRobot.idlでインターフェースを定義済み
  - これをインポートすればコンシューマ側は簡単に作成できる

MobileRobot.idl

- 移動ロボットナビゲーションに関わるインターフェースやデータ型を定義
- OpenRTM-aist同梱のExtendedDataTypes.idlやInterfaceDataTypes.idlを再利用
- 統一されたインターフェース設計思想
  - 値渡しでデータを渡し，参照渡しでデータを受け取る．返り値で処理の成否を受け取る

IDLファイル

```idl
#include "BasicDataType.idl"
#include "ExtendedDataTypes.idl"
#include "InterfaceDataTypes.idl"

module RTC {

  /*!
   * @struct OCMap
   * @brief OccupancyGridMap Data
   */
  struct OGMap
  {
    /// Time stamp.
    Time tm;
    /// OccupancyGridMap Configuration
    OGMapConfig config;
    /// OccupancyGridMap Data
    OGMapTile map;
  };
```

MobileRobot.idl

OGMapはマップデータ

OGMapConfig型や

OGMapTile型は

InterfaceDataTypes.idl

で定義されている

```idl
/*!
 * @struct OGMapConfig
 * @brief Configuration of a occupancy-grip map.
 */
struct OGMapConfig
{
    /// Scale on the x axis (metres per cell).
    double xScale;
    /// Scale on the y axis (metres per cell).
    double yScale;
    /// Number of cells along the x axis.
    unsigned long width;
    /// Number of cells along the y axis.
    unsigned long height;
    /// Pose of the cell at (0, 0) in the real world.
    Pose2D origin;
};

/*!
 * @typedef OGMapCells
 */
typedef sequence<octet> OGMapCells;

/*!
 * @struct OGMapTile
 * @brief A tile from an occupancy-grid map.
 */
struct OGMapTile
{
    /// X coordinate of the (0, 0) cell of this tile in the whole map.
    unsigned long column;
    /// Y coordinate of the (0, 0) cell of this tile in the whole map.
    unsigned long row;
    /// Number of cells along the x axis in this tile;
    unsigned long width;
    /// Number of cells along the y axis in this tile;
    unsigned long height;
    /// Tile cells in (row, column) order
    OGMapCells cells;
};
```

InterfaceDataTypes.idl

IDLファイル

```
enum RETURN_VALUE {
  // COMMON
  RETVAL_OK,
  RETVAL_INVALID_PARAMETER,
  RETVAL_EMPTY_MAP,
  RETVAL_INVALID_PRECONDITION,
  RETVAL_NOT_IMPL,
  RETVAL_UNKNOWN_ERROR,
  RETVAL_NOT_FOUND,

  RETVAL_ODOMETRY_INVALID_VALUE,
  RETVAL_ODOMETRY_TIME_OUT,
  RETVAL_RANGE_INVALID_VALUE,
  RETVAL_RANGE_TIME_OUT,

  RETVAL_EMERGENCY_STOP,
  RETVAL_OUTOF_RANGE

};


enum MAPPER_STATE {
  MAPPER_STOPPED,
  MAPPER_MAPPING,
  MAPPER_SUSPEND,
  MAPPER_ERROR,
  MAPPER_UNKNOWN
};

enum FOLLOWER_STATE {
  FOLLOWER_STOPPED,
  FOLLOWER_FOLLOWING,
  FOLLOWER_SUSPEND,
  FOLLOWER_ERROR,
  FOLLOWER_UNKNOWN
};
```

MobileRobot.idl

enumで返り値の基本値（RETURN_VALUE）や，

MapperやFollowerの状態み関する変数が定義されている

IDLファイル

```
/*!
 * @interface OGMapper
 * @brief Occupancy Grid Map Builder Service Interface
 */
interface OGMapper {

  /// Initialize Current Build Map Data
  RETURN_VALUE initializeMap(in OGMapConfig config, in Pose2D initialPose);

  RETURN_VALUE startMapping();

  RETURN_VALUE stopMapping();

  RETURN_VALUE suspendMapping();

  RETURN_VALUE resumeMapping();

  RETURN_VALUE getState(out MAPPER_STATE state);

  /// Request Current Build Map Data
  RETURN_VALUE requestCurrentBuiltMap(out OGMap map);
};/*!
  * @interface OGMapServer
  * @brief Occupancy Grid Map Service Interface
  */
 interface OGMapServer {

   /// Request Current Build Map Data
   RETURN_VALUE requestCurrentBuiltMap(out OGMap map);
 };
```

OGMapperはマップ作成用インターフェース

startMappingでマップ作成開始

stopMappingでマップ作成終了

requestCurrentBuiltMapで現在のマップ要求

OGMapServerはマップを単純に配信するための
インターフェース

requestCurrentBuiltMapでマップを取得

IDLファイル

```
interface PathPlanner {
  /// Plan Path from PathPlanParater.
    RETURN_VALUE planPath(in PathPlanParameter param, out Path2D outPath);
};
struct PathPlanParameter {
  //Environmental Map
  OGMap map;
  /// Location of the goal.
  Pose2D targetPose;
  /// Location of Robot.
  Pose2D currentPose;
  /// How far away from the waypoint is considered success (radius in metres).
  double distanceTolerance;
  /// How much off the target heading is considered success (in radians).
  double headingTolerance;
  /// Target time to arrive at the waypoint by.
  Time timeLimit;
  /// Maximum sped to travel at while heading to the waypoint.
  Velocity2D maxSpeed;
};
```

PathPlannerは軌道計画のためのインターフェース

planPathで軌道計画をする

PathPlanParameterは，PathPlannerに渡すためのパラメータ構造体

```
interface PathFollower {
  RETURN_VALUE followPath(in Path2D path);

  RETURN_VALUE getState(out FOLLOWER_STATE state);

  RETURN_VALUE followPathNonBlock(in Path2D path);
};
```

PathFollowerは軌道追従のためのインターフェース

IDLファイル

```
/*!
 * @struct Waypoint2D
 * @brief A waypoint in 2D space, including constraints.
 */
struct Waypoint2D
{
    /// Location of the waypoint.
    Pose2D target;
    /// How far away from the waypoint is considered success (radius in metres).
    double distanceTolerance;
    /// How much off the target heading is considered success (in radians).
    double headingTolerance;
    /// Target time to arrive at the waypoint by.
    Time timeLimit;
    /// Maximum sped to travel at while heading to the waypoint.
    Velocity2D maxSpeed;
};

/*!
 * @typedef Waypoint2DList
 */
typedef sequence<Waypoint2D> Waypoint2DList;

/*!
 * @struct Path2D
 * @brief A time-stamped path in 2D space.
 */
struct Path2D
{
    /// Time stamp.
    Time tm;
    /// The sequence of waypoints that make up the path.
    Waypoint2DList waypoints;
};
```

Path2DはWaypoint2Dの配列

Waypointは，目標位置姿勢，

距離・角度許容差，

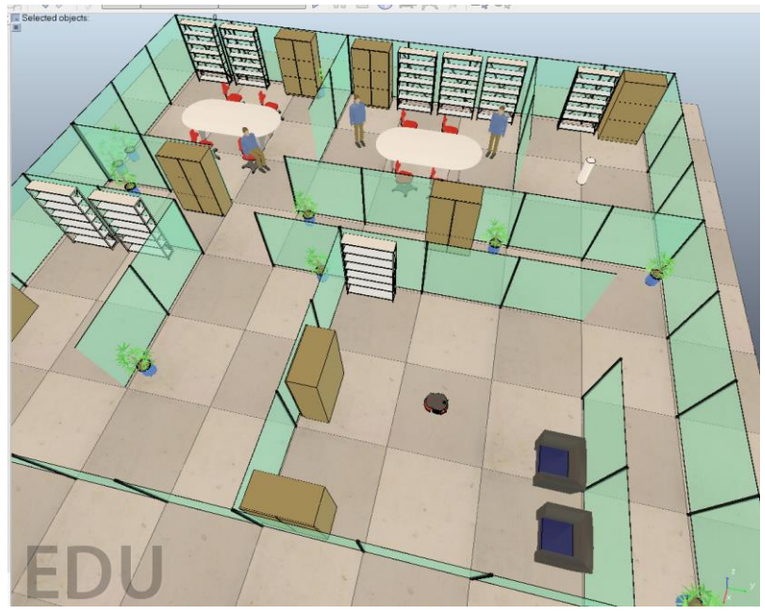
タイムリミットや最大速度が入る

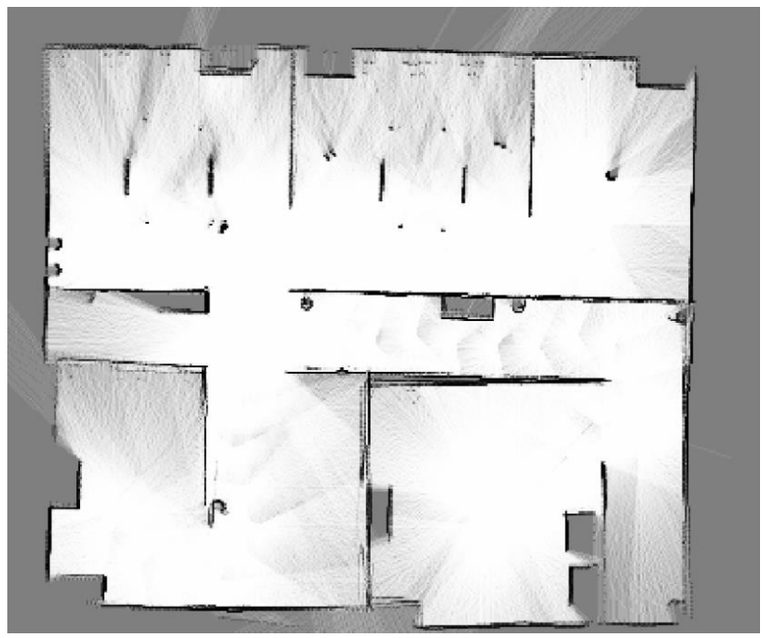
IDLファイル全文 (MobileRobot.idl)

```
#include "BasicDataType.idl"
#include "ExtendedDataTypes.idl"
#include "InterfaceDataTypes.idl"

module RTC {

  /*!
   * @struct OCMap
   * @brief OccupancyGridMap Data
   */
  struct OGMap
  {
    /// Time stamp.
    Time tm;
    /// OccupancyGridMap Configuration
    OGMapConfig config;
    /// OccupancyGridMap Data
    OGMapTile map;
  };

  enum RETURN_VALUE {
      // COMMON
      RETVAL_OK,
      RETVAL_INVALID_PARAMETER,
      RETVAL_EMPTY_MAP,
      RETVAL_INVALID_PRECONDITION,
      RETVAL_NOT_IMPL,
      RETVAL_UNKNOWN_ERROR,
      RETVAL_NOT_FOUND,

      RETVAL_ODOMETRY_INVALID_VALUE,
      RETVAL_ODOMETRY_TIME_OUT,
      RETVAL_RANGE_INVALID_VALUE,
      RETVAL_RANGE_TIME_OUT,

      RETVAL_EMERGENCY_STOP,
      RETVAL_OUTOF_RANGE

  };


  enum MAPPER_STATE {
    MAPPER_STOPPED,
    MAPPER_MAPPING,
    MAPPER_SUSPEND,
    MAPPER_ERROR,
    MAPPER_UNKNOWN
  };

  enum FOLLOWER_STATE {
    FOLLOWER_STOPPED,
    FOLLOWER_FOLLOWING,
    FOLLOWER_SUSPEND,
    FOLLOWER_ERROR,
    FOLLOWER_UNKNOWN
  };

  /*!
   * @interface OGMapper
   * @brief Occupancy Grid Map Builder Service Interface
   */
  interface OGMapper {

    /// Initialize Current Build Map Data
    RETURN_VALUE initializeMap(in OGMapConfig config,
                               in Pose2D initialPose);

    RETURN_VALUE startMapping();

    RETURN_VALUE stopMapping();

    RETURN_VALUE suspendMapping();

    RETURN_VALUE resumeMapping();

    RETURN_VALUE getState(out MAPPER_STATE state);

    /// Request Current Build Map Data
    RETURN_VALUE requestCurrentBuiltMap(out OGMap map);
  };

  /*!
   * @interface OGMapServer
   * @brief Occupancy Grid Map Service Interface
   */
  interface OGMapServer {

    /// Request Current Build Map Data
    RETURN_VALUE requestCurrentBuiltMap(out OGMap map);
  };

  struct PathPlanParameter {
      //Environmental Map
      OGMap map;
    /// Location of the goal.
    Pose2D targetPose;
    /// Location of Robot.
    Pose2D currentPose;
    /// How far away from the waypoint is
    /// considered success (radius in metres).
    double distanceTolerance;
    /// How much off the target heading
    /// is considered success (in radians).
   double headingTolerance;
    /// Target time to arrive at the waypoint by.
    Time timeLimit;
    /// Maximum sped to travel
    /// at while heading to the waypoint.
    Velocity2D maxSpeed;
  };

  struct PathPlanParameter {
      //Environmental Map
      OGMap map;
    /// Location of the goal.
    Pose2D targetPose;
    /// Location of Robot.
    Pose2D currentPose;
    /// How far away from the waypoint is considered
    /// success (radius in metres).
    double distanceTolerance;
    /// How much off the target heading is considered
    /// success (in radians).
    double headingTolerance;
    /// Target time to arrive at the waypoint by.
    Time timeLimit;
    /// Maximum sped to travel at while heading to the waypoin
    Velocity2D maxSpeed;
  };

  interface PathPlanner {
    /// Plan Path from PathPlanParater.
        RETURN_VALUE planPath(in PathPlanParameter param,
                              out Path2D outPath);
  };

  interface PathFollower {
    RETURN_VALUE followPath(in Path2D path);

    RETURN_VALUE getState(out FOLLOWER_STATE state);

    RETURN_VALUE followPathNonBlock(in Path2D path);
  };

};

#endif
```

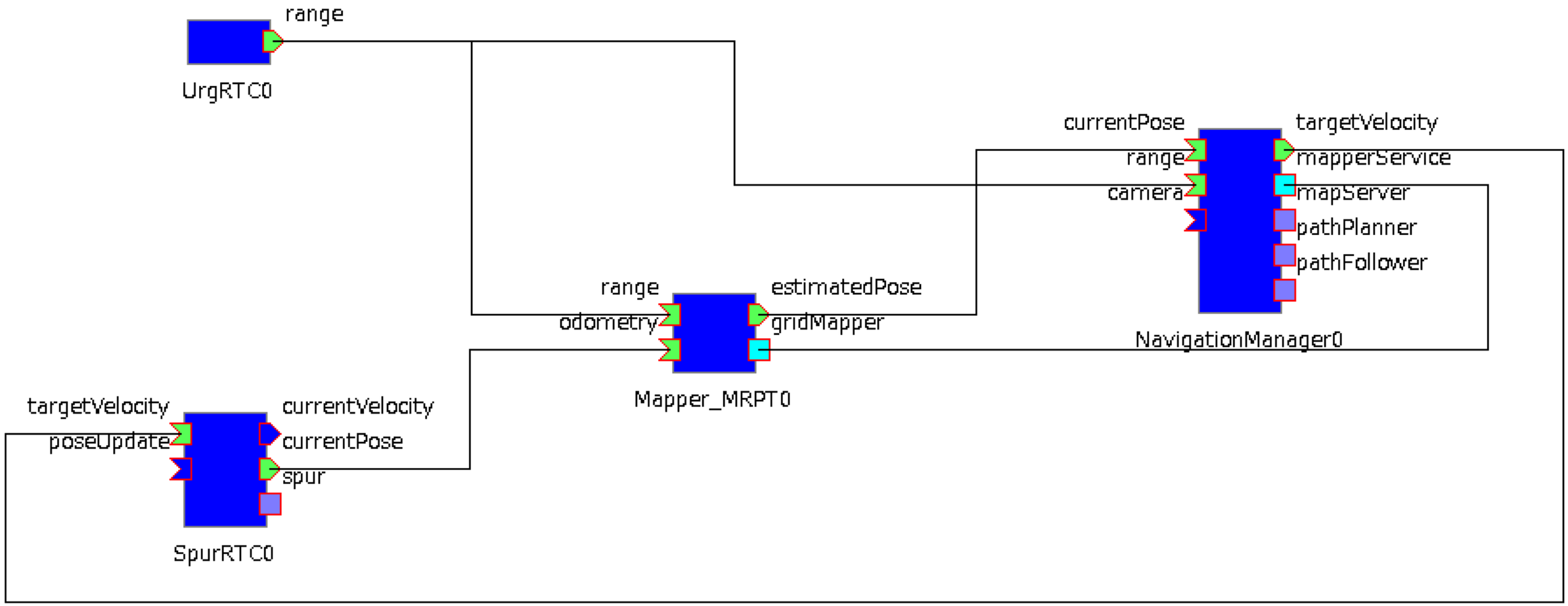
マップの作成

- ロボットに北陽URG等のレーザースキャナを搭載
- ロボットの移動情報 (オドメトリ) とスキャン情報からマップを作成
- ロボットの移動はジョイスティックなどで指令を送る



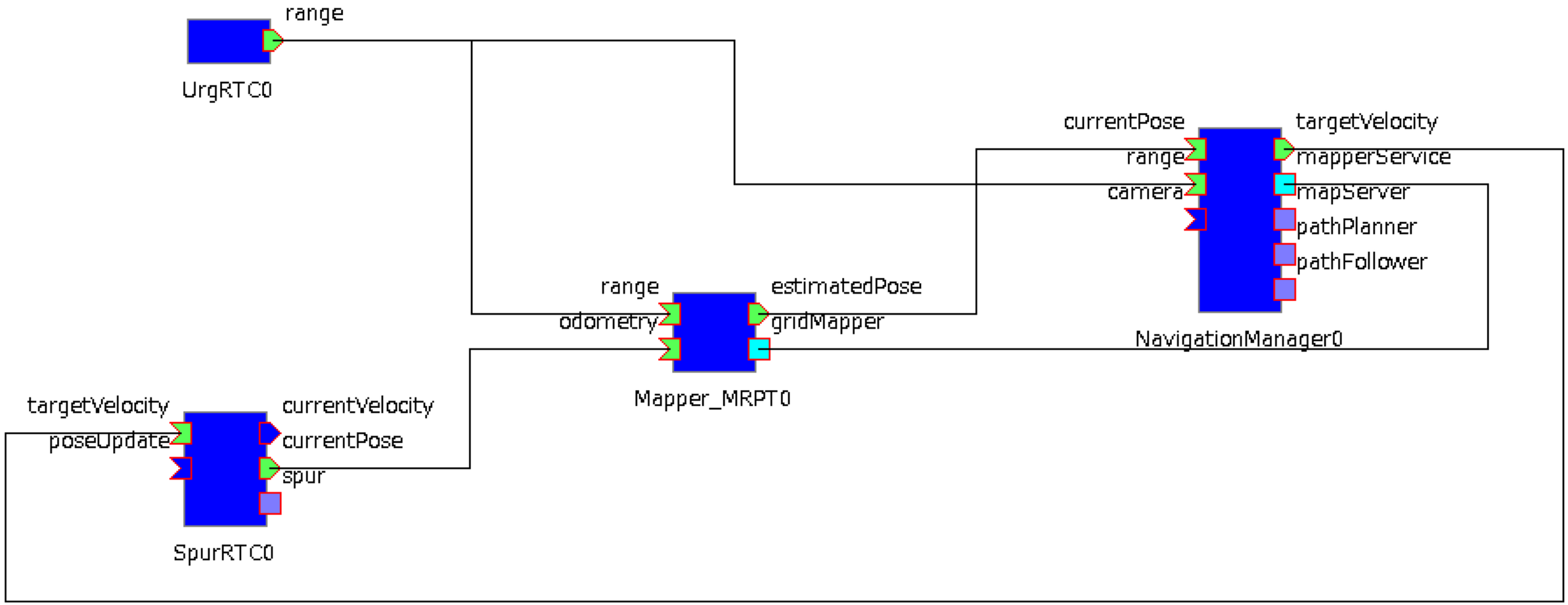


マップ作成用フレームワーク
(基本的な使い方)



全RTC起動し図のように接続

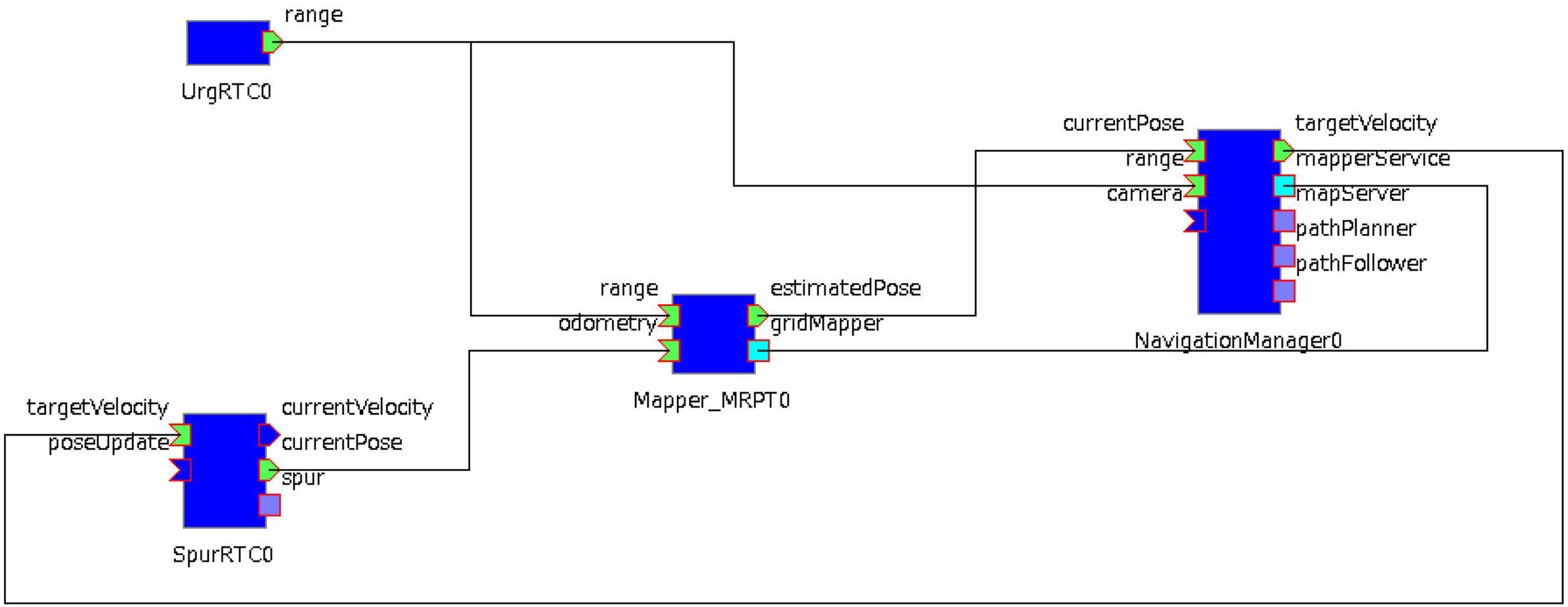
- Mapper_ALL.batで起動



コンフィグレーションの設定

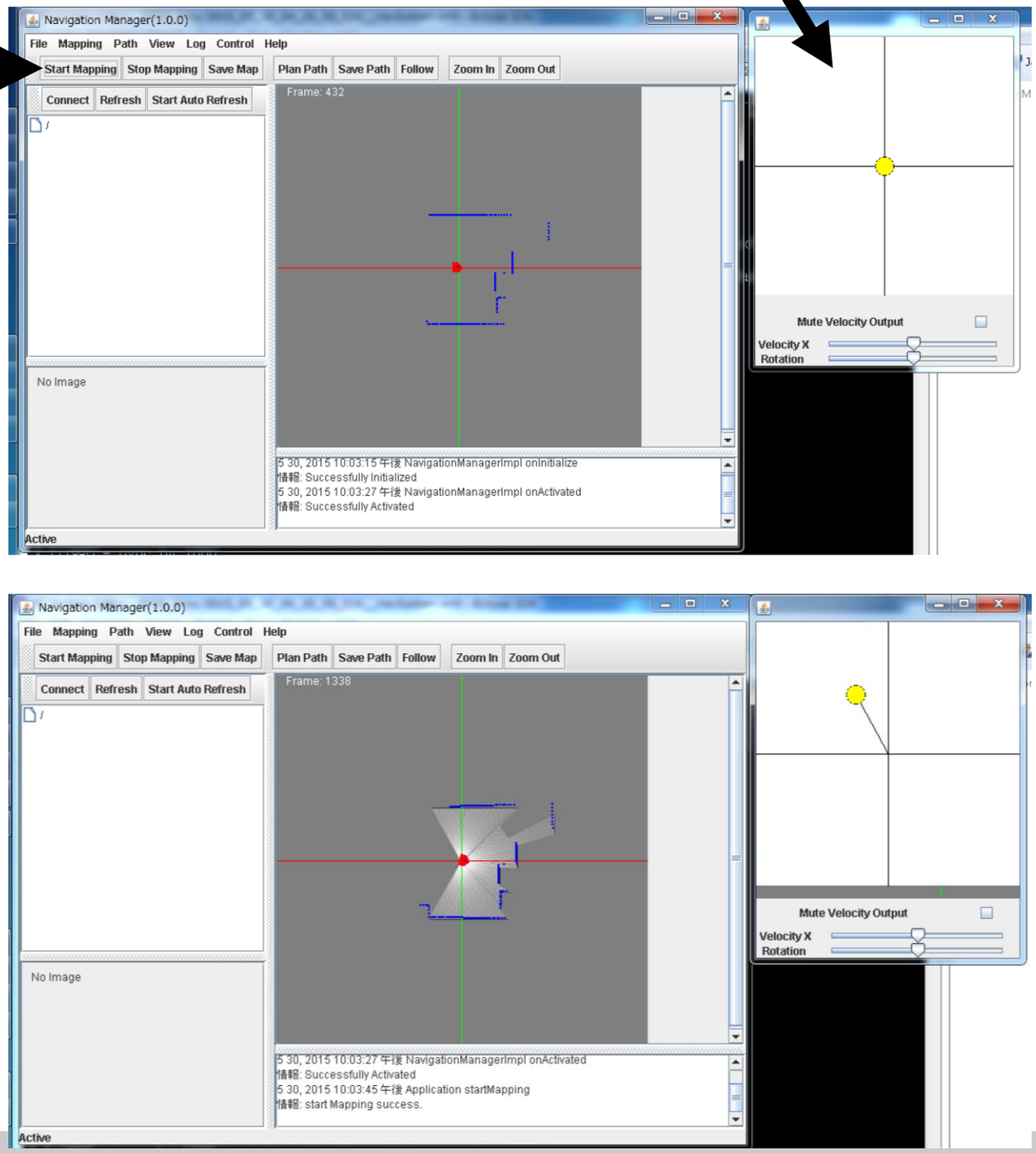
- UrgRTCのCOMポートをコンフィグレーション (port_name) で指定
- geometry_x, y, zで，ロボットの座標系 (車軸中心) からのオフセットを指定 (単位[m])

全RTC起動をアクティブ化



GUIの利用

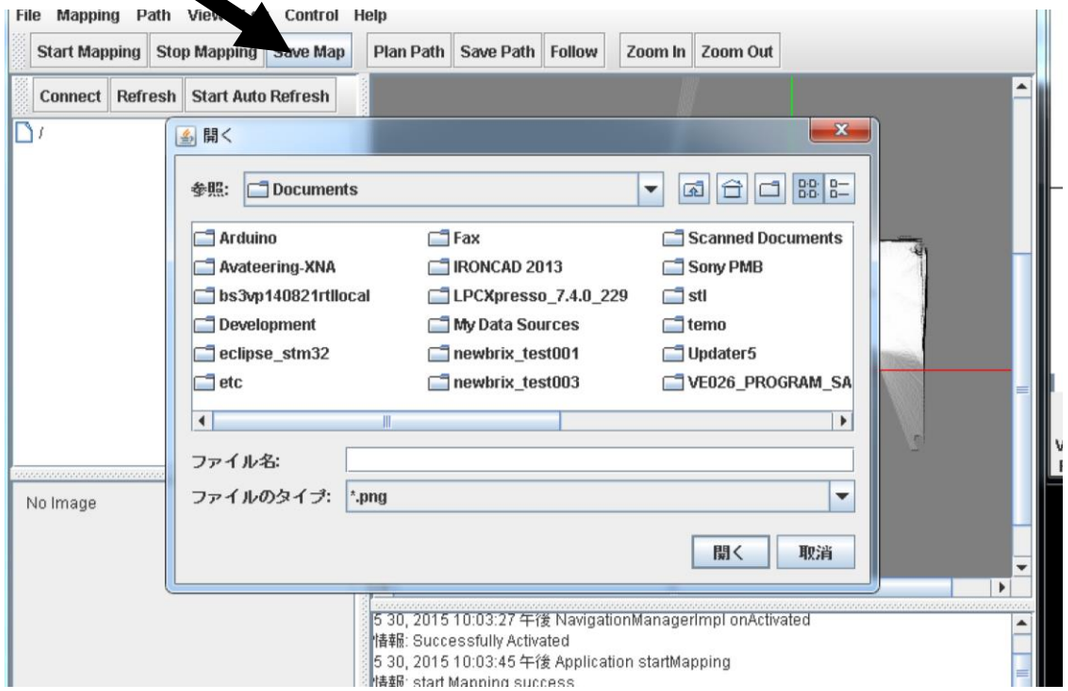
- Start Mappingボタンでマッピング開始
- NavigationManagerのtargetVelocityポートを接続していれば，右図のGUIジョイスティックが起動
- ジョイスティック (黄色い円) を上に引っ張ると，ロボットが前進
- 別のジョイスティックRTCを使うことも可能



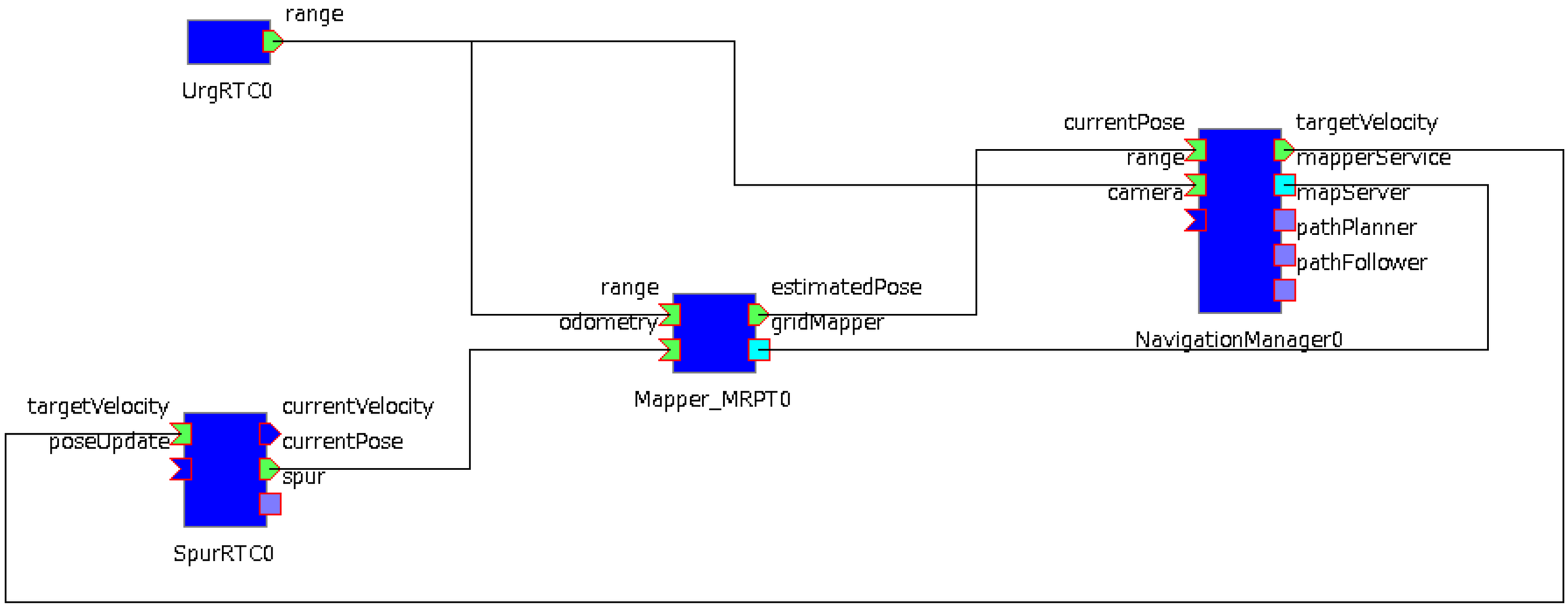
GUIの利用

Saveボタン

- Save Mapでマップを保存
- マップデータは二つのファイルで保存される
  - ****.png：グリッドマップのデータ
  - ****.yaml：マップの原点やピクセル/長さの比の情報

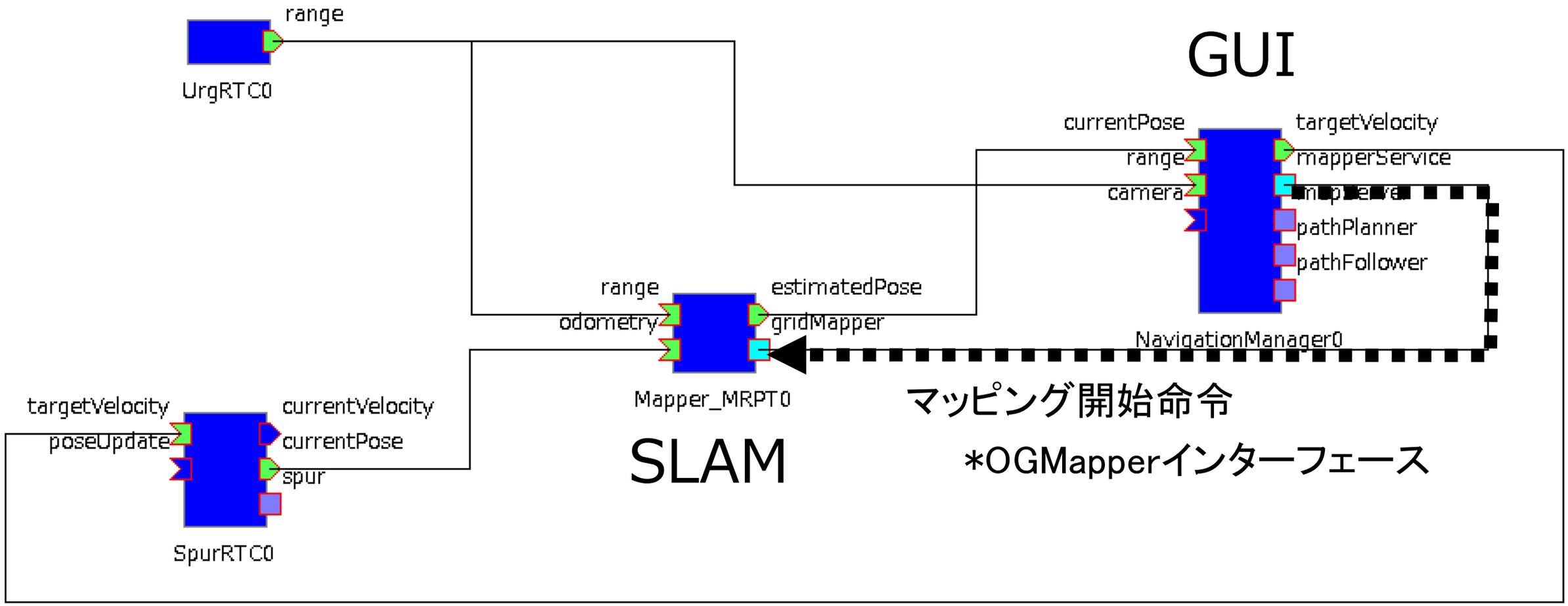


マップ作成用フレームワーク (操作とRTシステムとの関係)



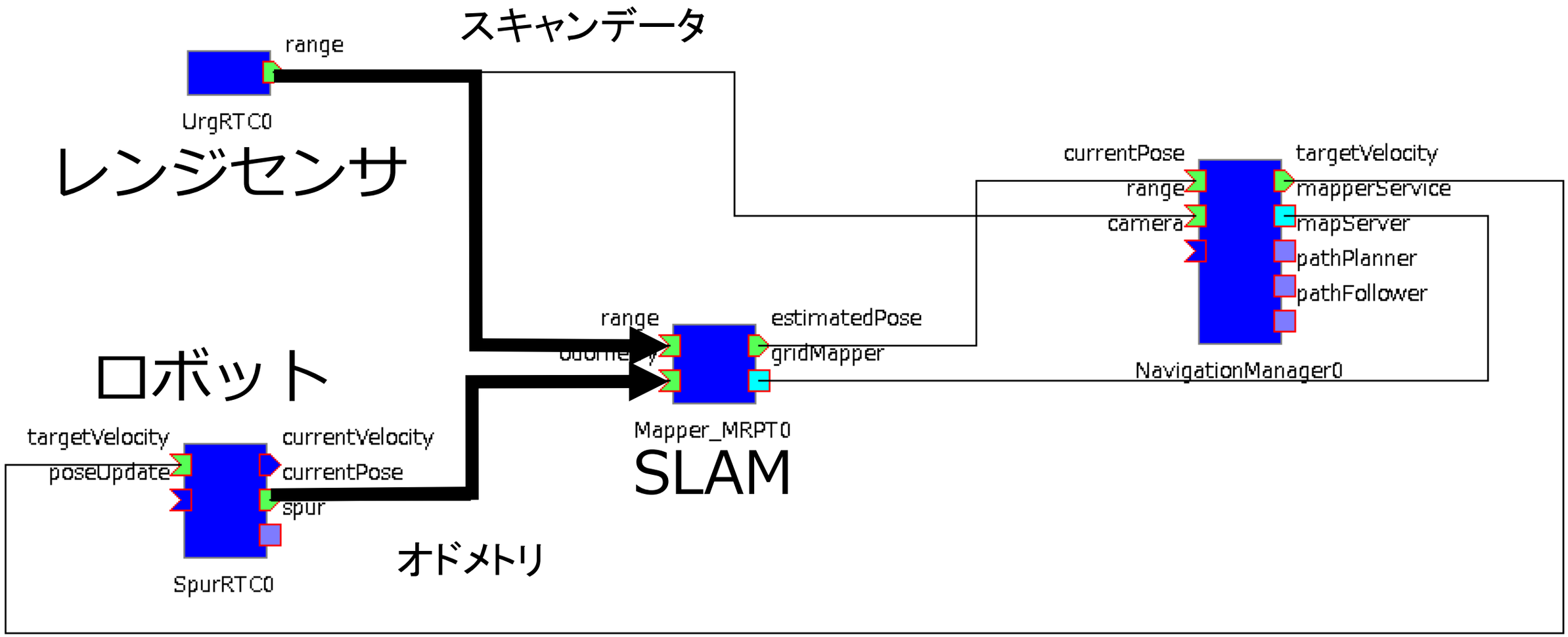
Start Mapping時

- GUIからSLAMモジュールにマッピング開始を指示



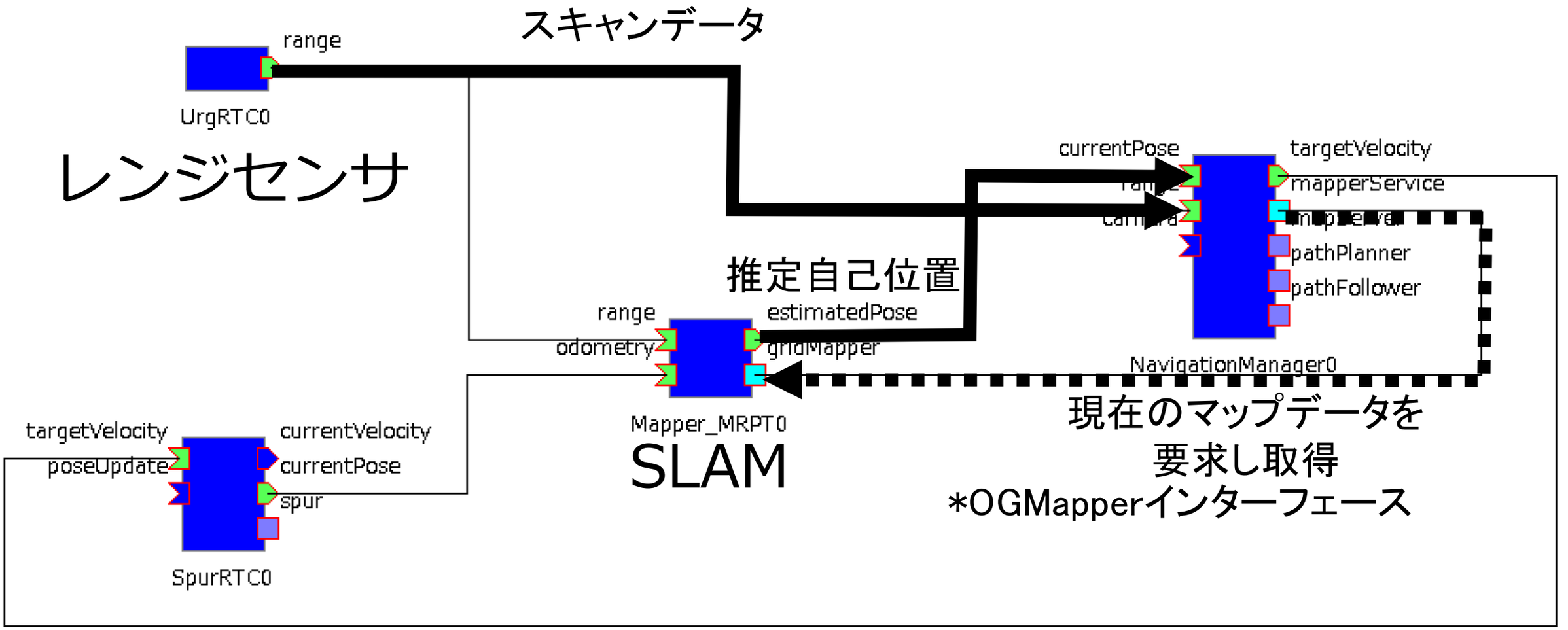
Mapping中

- レンジスキャナのデータとロボットのオドメトリからマップを作成



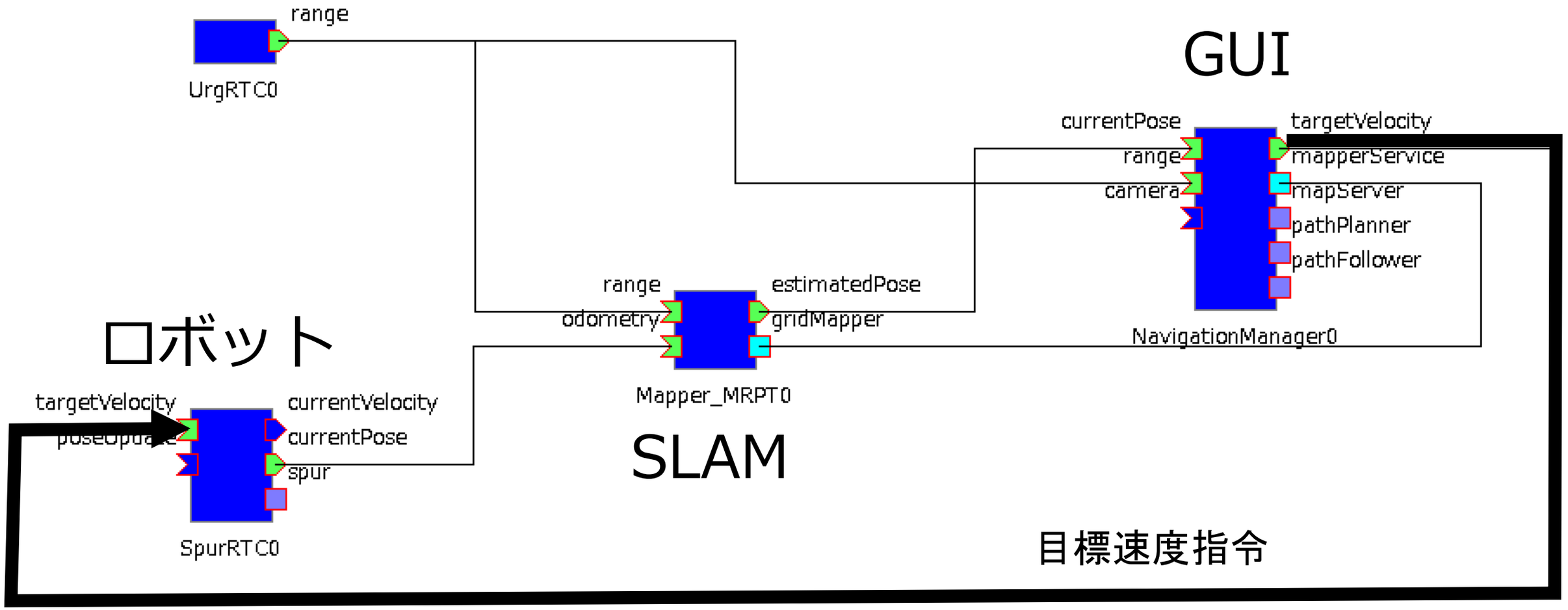
Mapping中の表示

- ロボットの自己位置とスキャンデータをデータポートから取得
- サービスポートで現在のマップデータを取得．重ねて表示



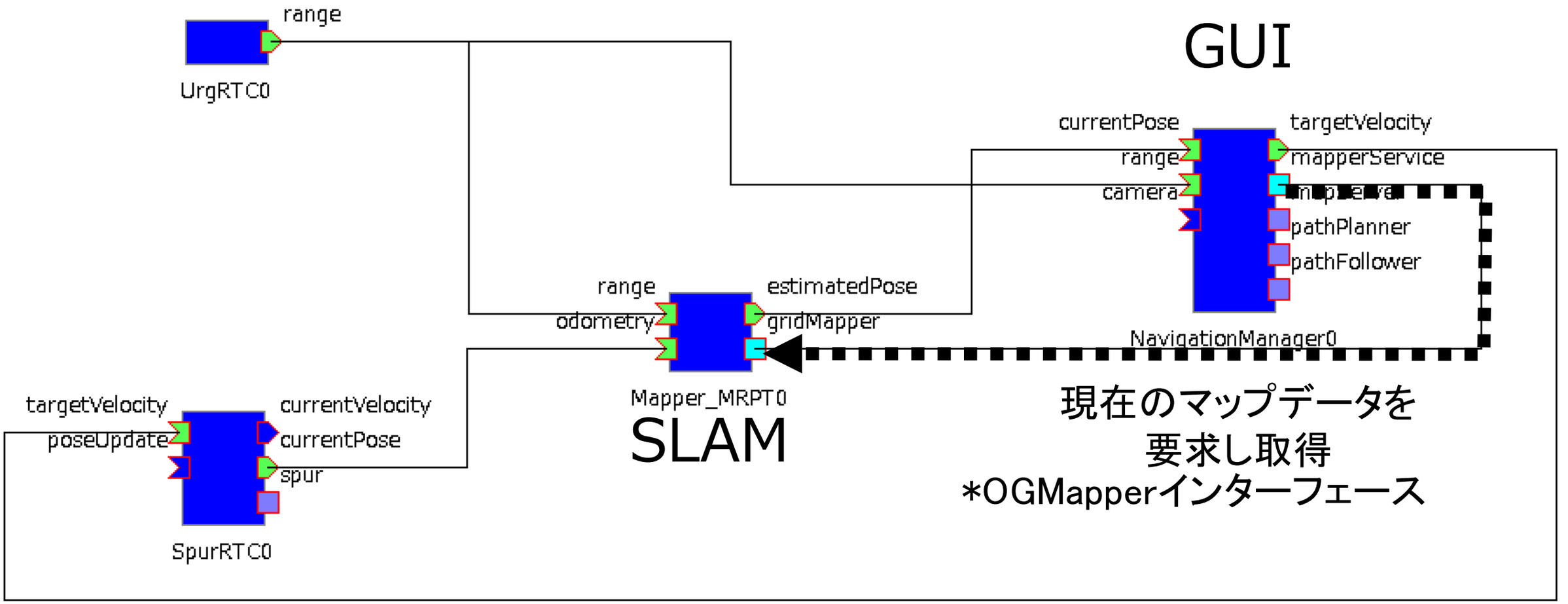
GUIジョイスティックでの操作

- targetVelocityをGUIから指示

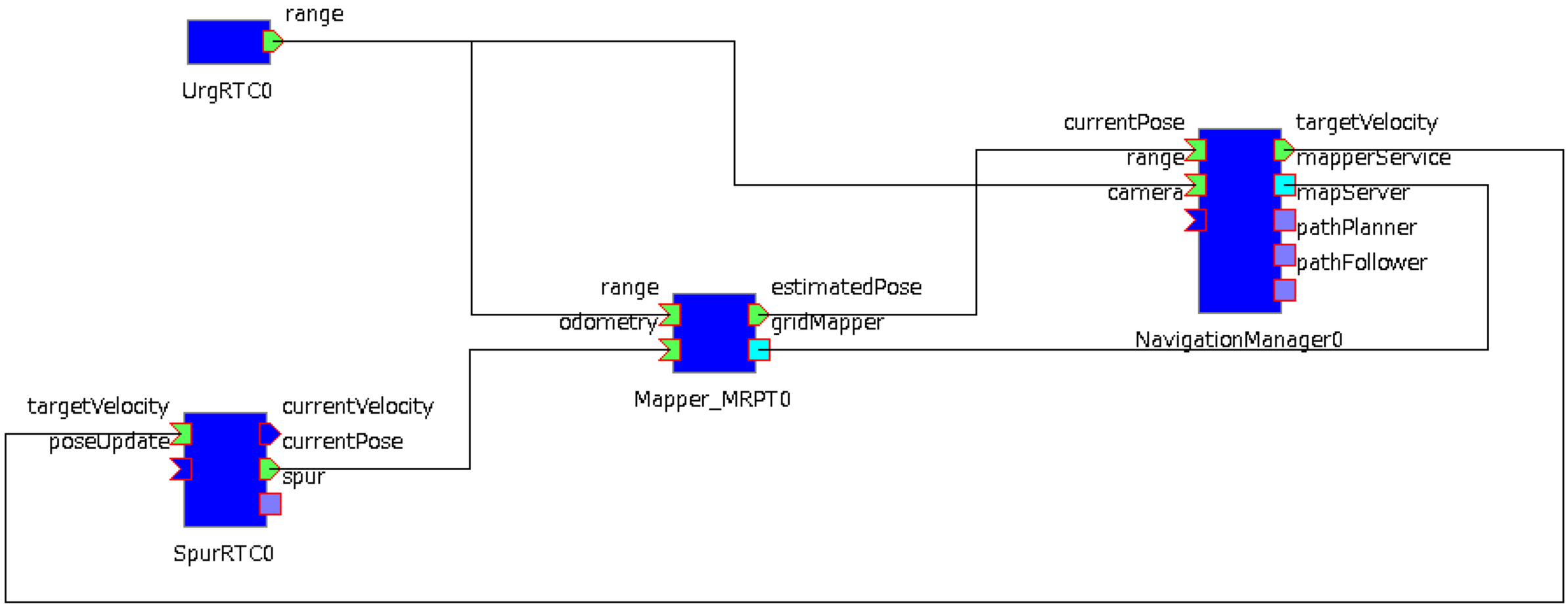


Save Map時

- サービスポートで現在のマップデータを取得．保存．



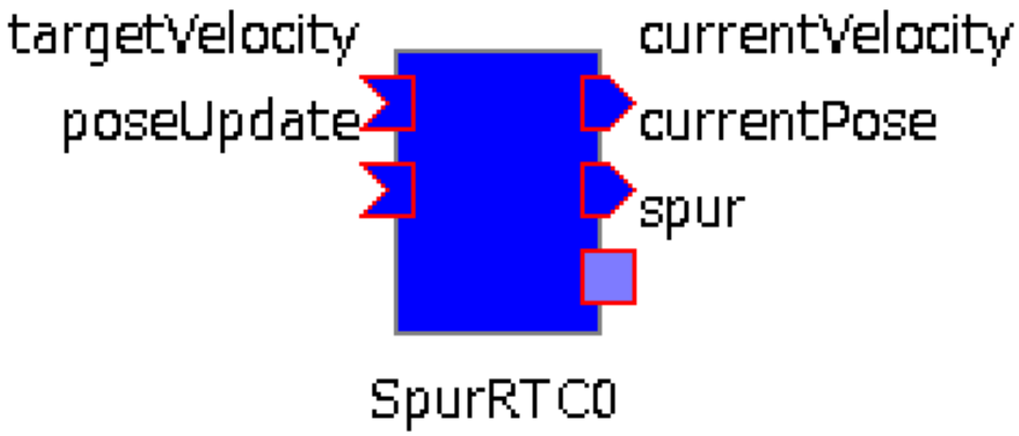
マップ作成用フレームワーク (各RTCの解説)



北陽電機 URG

- レーザーセンサ URGのRTC
- 出力ポート
  - range：RangeData
    - レーザースキャナ出力
- 主なコンフィグレーション
  - スキャナのロボット座標系での位置，オフセット [m]
    - geometry_x
    - geometry_y
    - geometry_z
  - COMポート番号 (COM3とか)
    - port_name

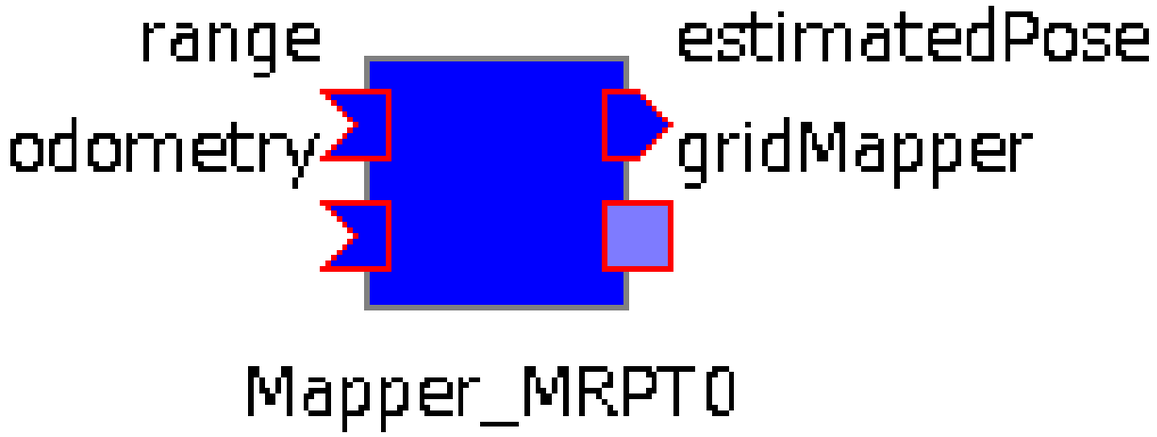
YPSpur対応RTC



- YPSpur対応RTコンポーネント
- 入力ポート
  - targetVelocity：TimedVelocity2D
    - 目標速度
  - poseUpdate：TimedPose2D
    - 現在位置の更新
- 出力ポート
  - currentVelocity：TimedVelocity2D
    - 現在速度
  - currentPose：TimedPose2D
    - 推定自己位置 (オドメトリ)

- 主なコンフィグレーション
  - 最大加速度 (activate時に更新) [m/s^2]
    - max_acc
    - max_rot_acc
  - 最大速度 (activate時に更新) [m/s]
    - max_vel
    - max_rot_vel

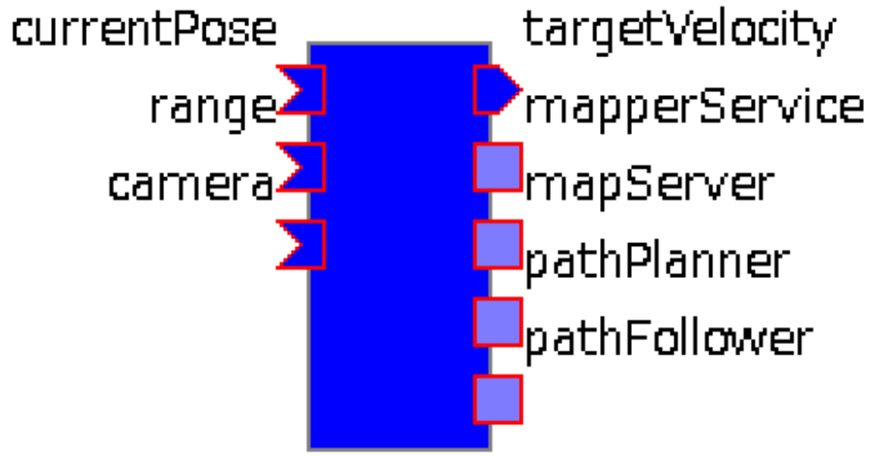
Mapper_MRPT



- SLAMモジュール (MRPTを利用)
- 入力ポート
  - range：RangeData
    - LiDAR入力
  - odometry：TimedPose2D
    - オドメトリ入力
- 出力ポート
  - estimatedPose：TimedPose2D
    - スキャンマッチ後の推定自己位置
- サービスポート
  - gridMapper：RTC::OGMapper (Provided)
    - マッピング開始・終了，マップ取得などの操作

- 主なコンフィグレーション
  - x_max, x_min：X軸方向のマップ広さ初期値 [m]
  - y_max, y_min：Y軸方向のマップ広さ初期値 [m]
  - resolution：グリッドの解像度 [m]

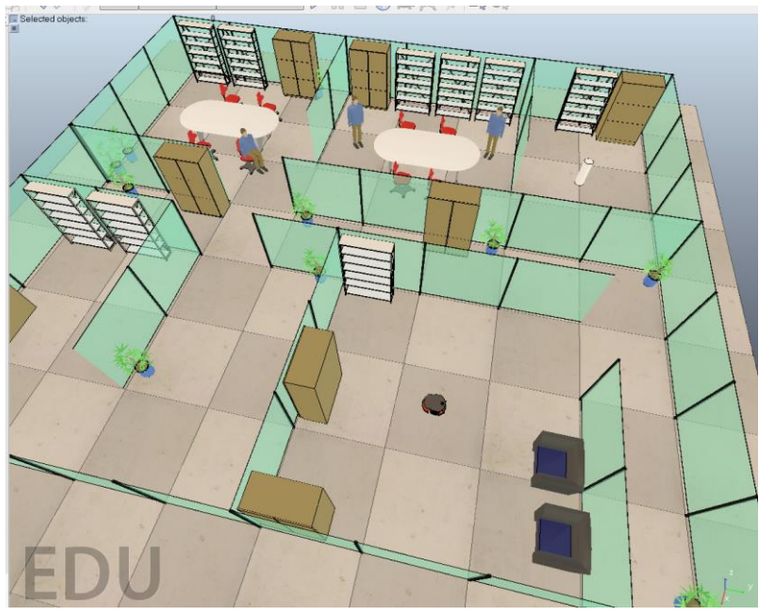
NavigationManager

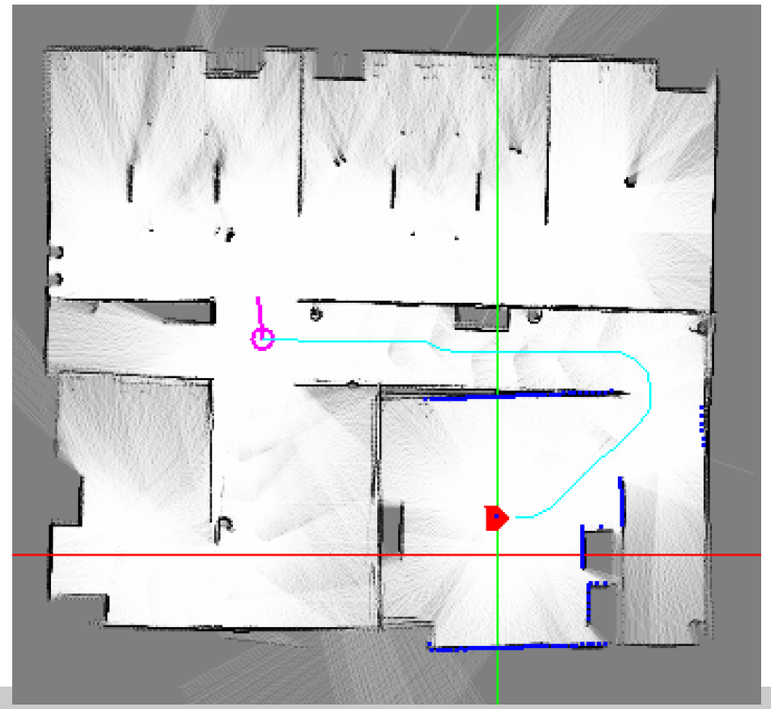


- ナビゲーション用GUI
- 入力ポート
  - currentPose：TimedPose2D
    - ロボットの自己位置 (表示用)
  - range：RangeData
    - LiDAR入力 (表示用)
  - camera：CameraImage
    - カメラ画像表示
- 出力ポート
  - targetVelocity：TimedVelocity2D
    - GUIジョイスティックでの操作時に接続
- サービスポート
  - mapperService：RTC::OGMapper (Required)
    - SLAMモジュールを接続．マッピングの開始・終了・マップ取得

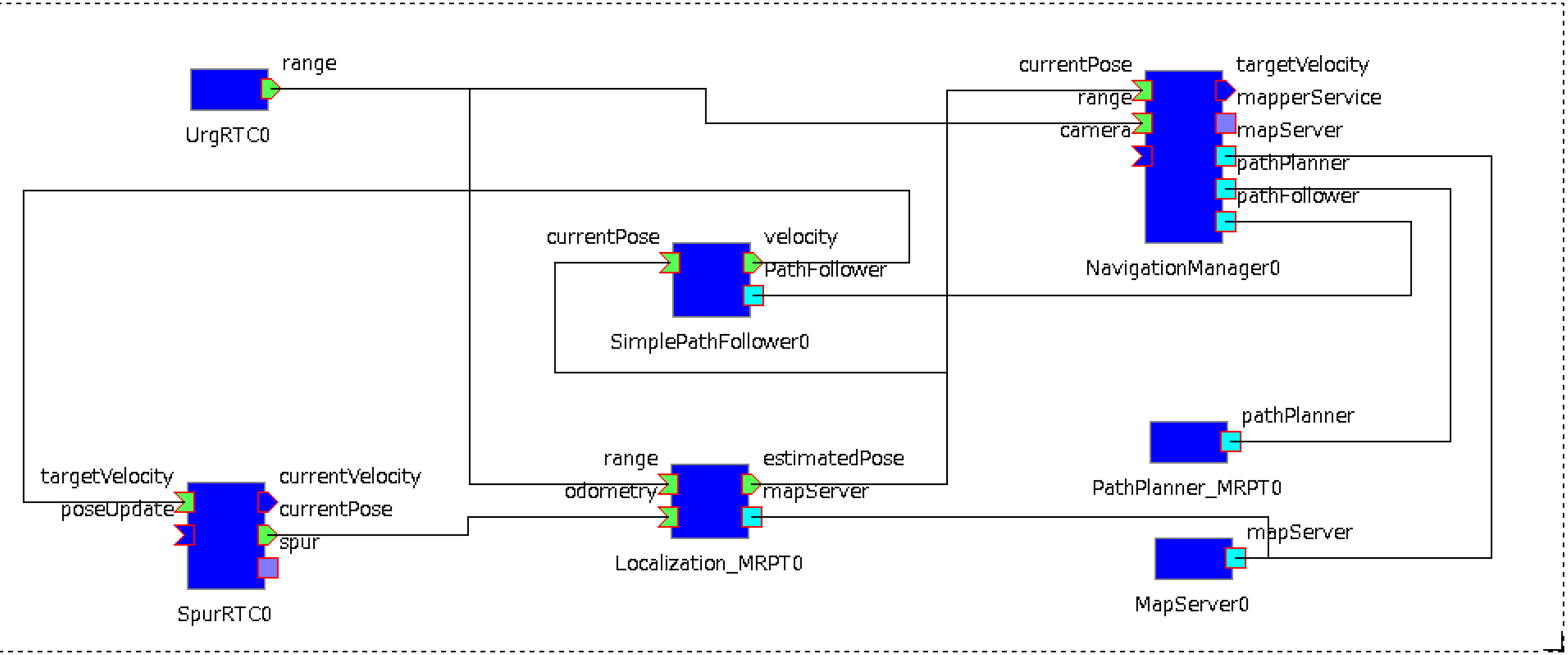
ナビゲーション

- ロボットに北陽URG等のレーザースキャナを搭載
- ロボットの移動情報 (オドメトリ) とスキャン情報をマップとマッチングして自己位置を高精度に推定
- マップ情報から軌道計画を自動化
- 計画した軌道への自動的な追従





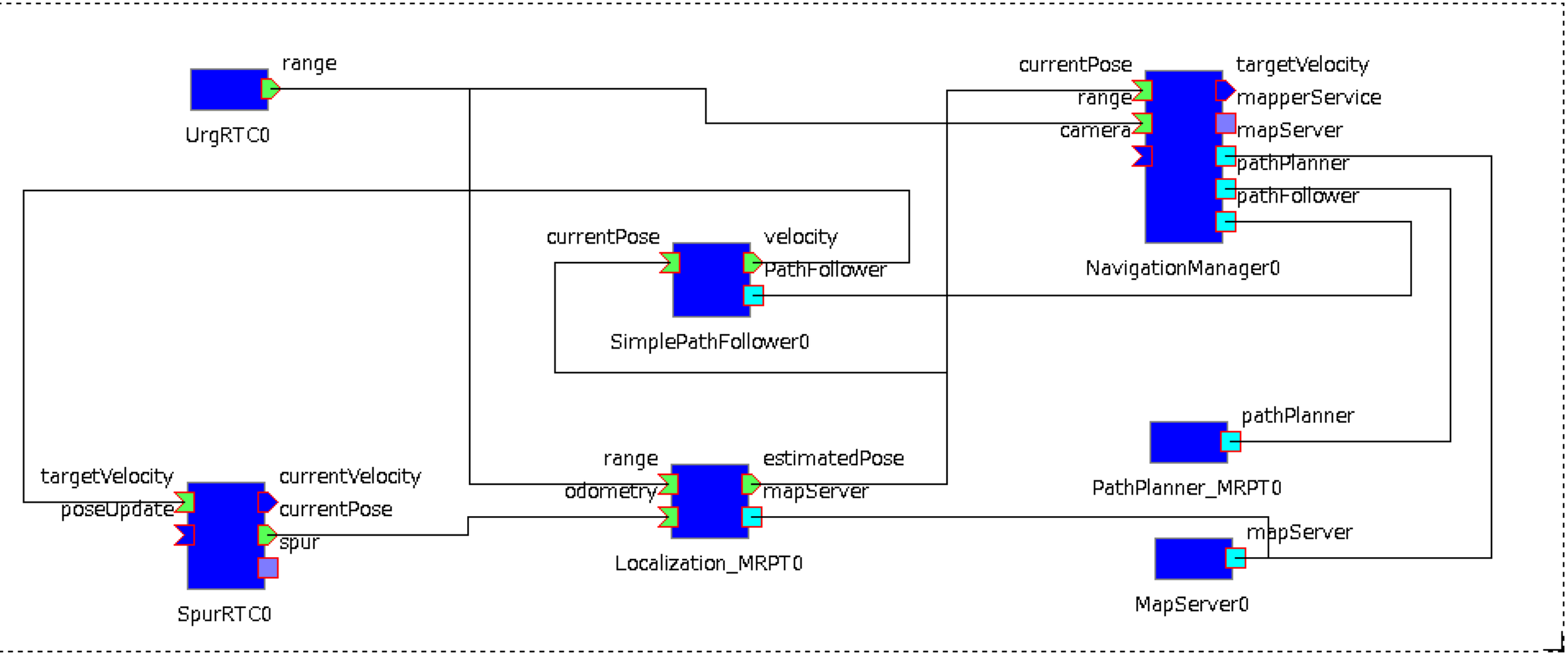
ナビゲーション用フレームワーク (基本的な使い方)



菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

全RTCを起動し図のように接続

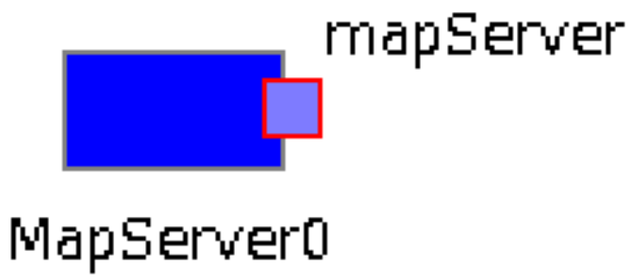
- Navi_ALL.batで起動



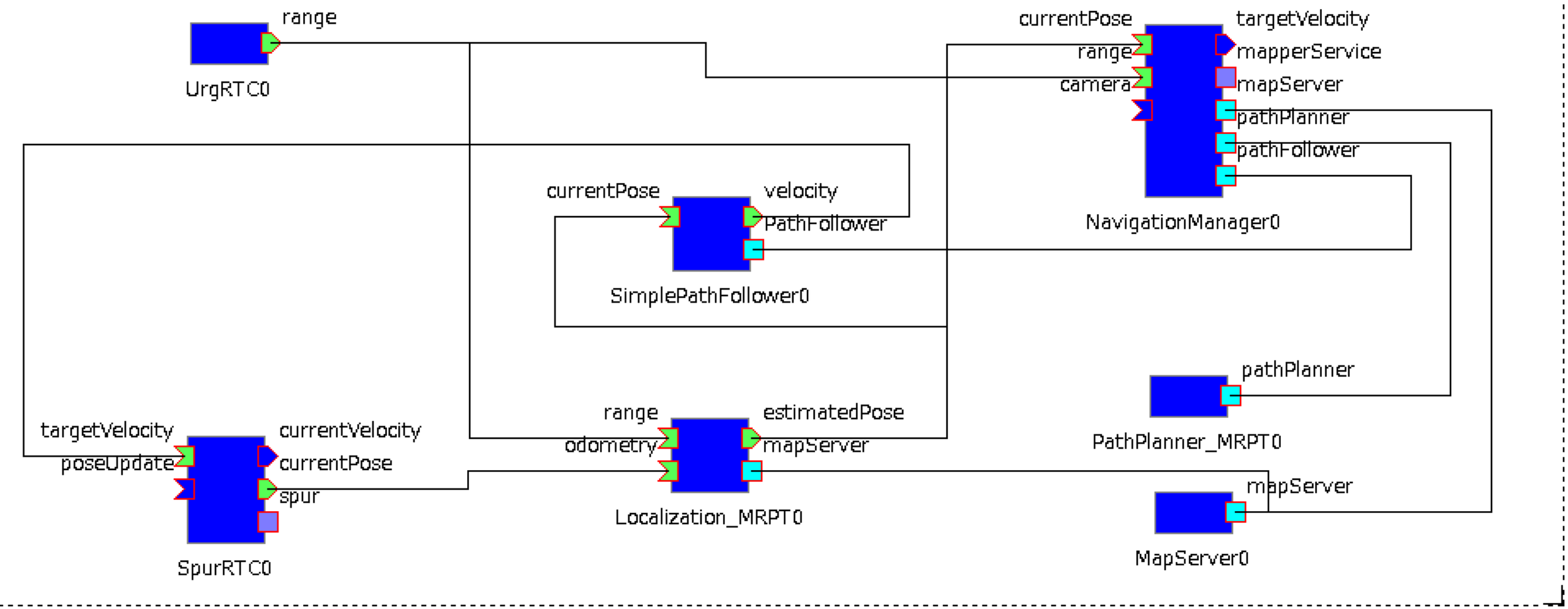
コンフィグレーションの設定

- UrgRTCのCOMポートをコンフィグレーション (port_name) で指定
- geometry_x, y, zで，ロボットの座標系 (車軸中心) からのオフセットを指定 (単位[m])

- MapServerのマップファイルのファイル名を設定
  - ****.yamlを設定すること

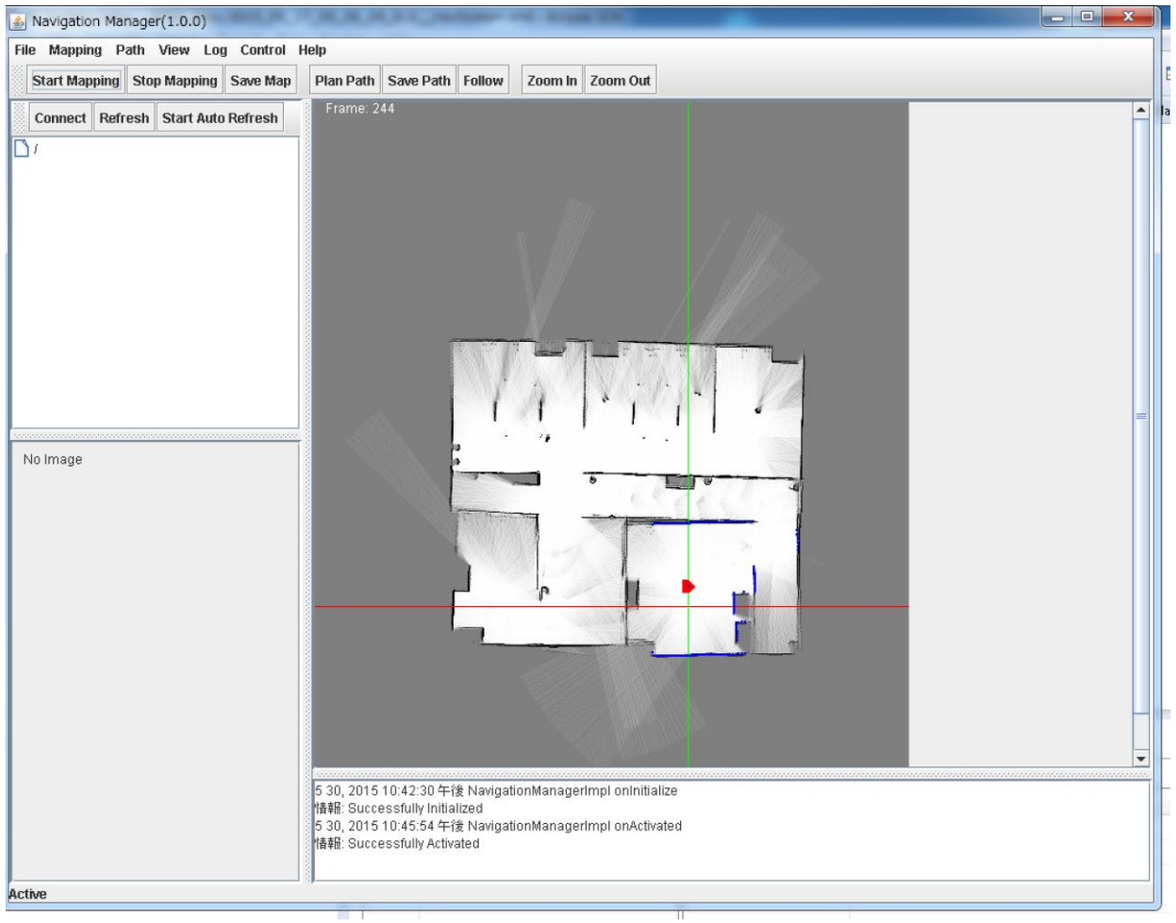


全RTCをActivate



GUIの利用

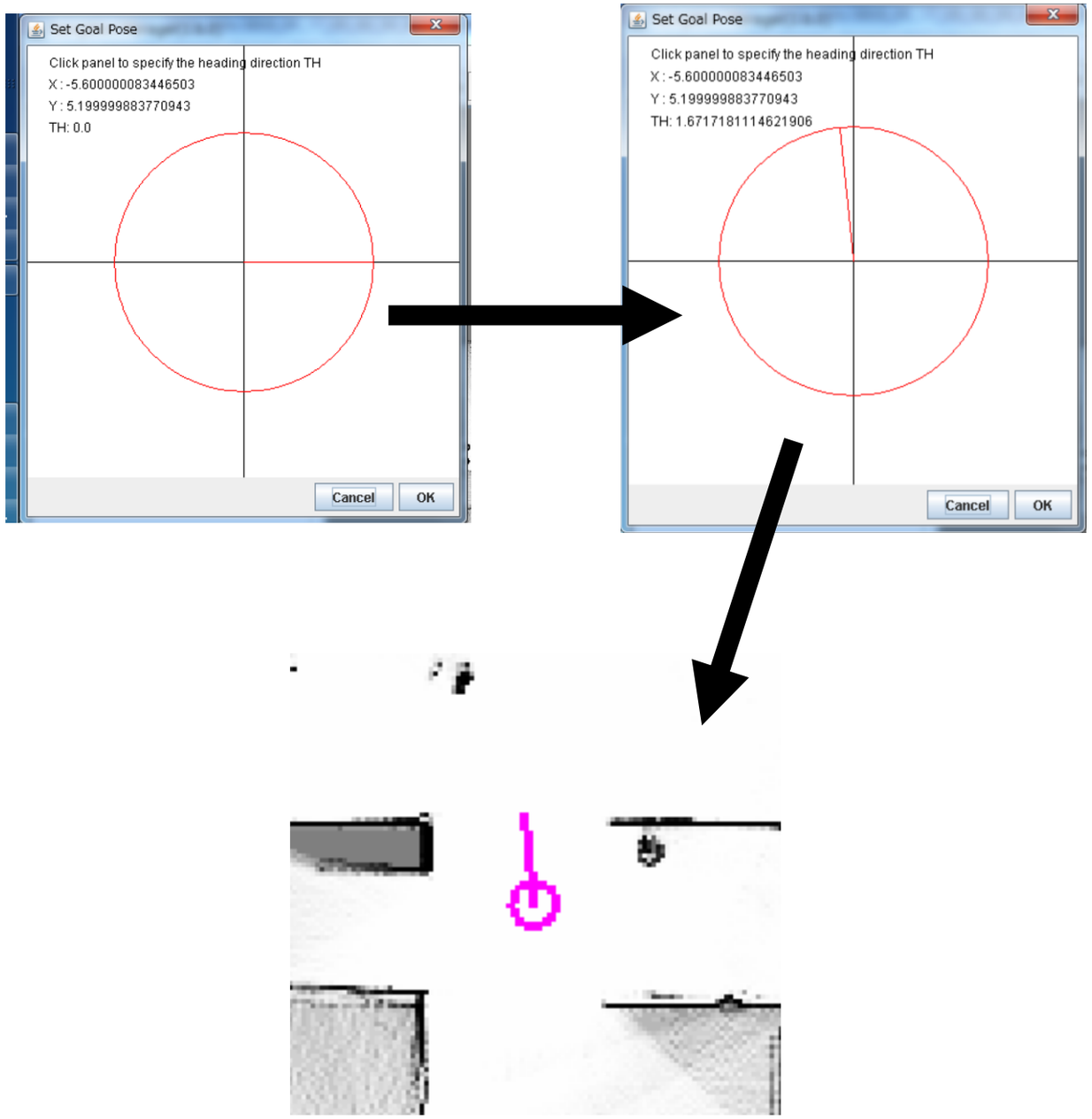
- MapServerにマップが読み込まれていれば，マップが表示され，ロボットの位置とレンジセンサのデータが表示される
- マップ上の目標位置をクリックするとゴールを指定できる



菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

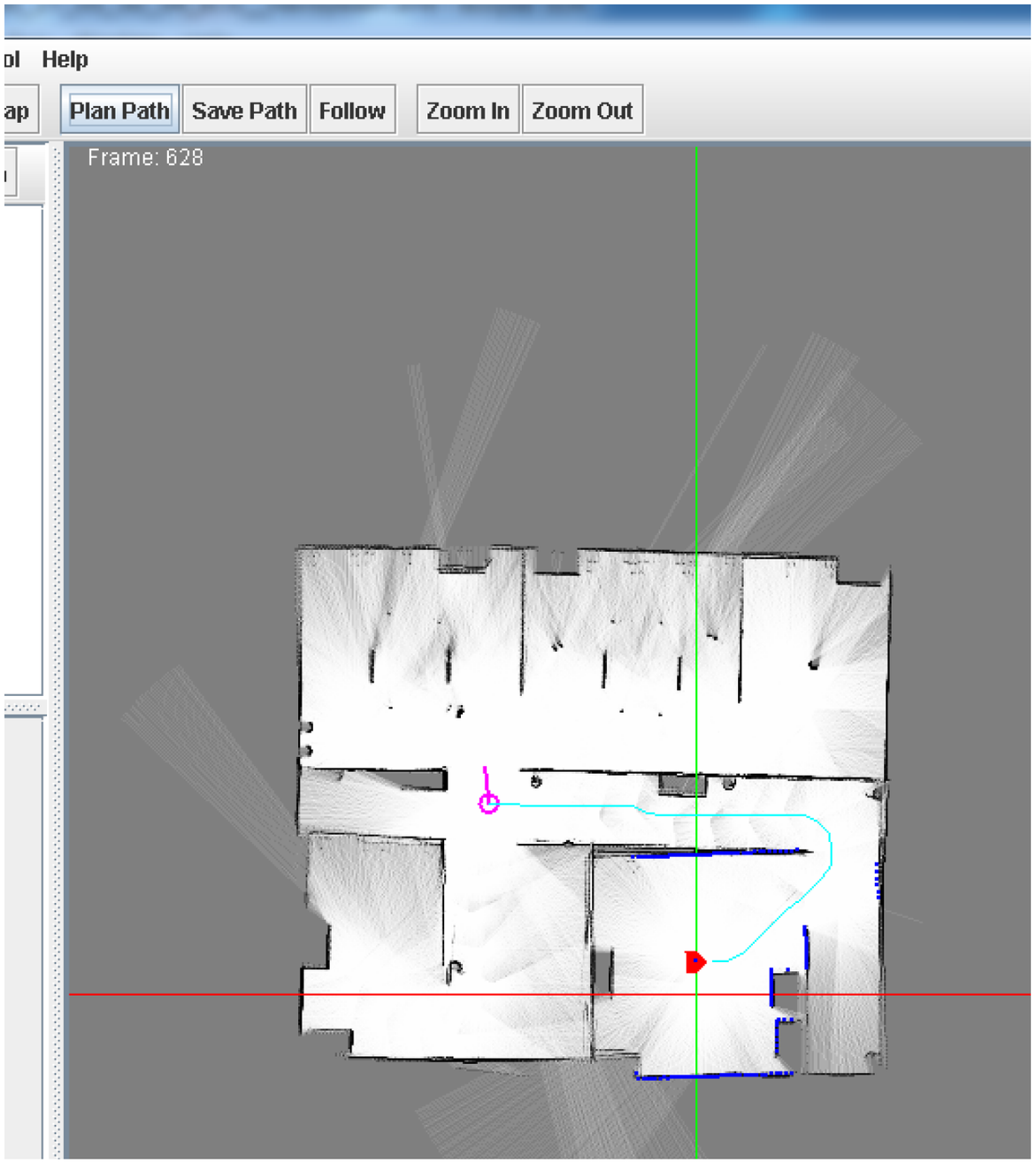
GUIの利用

- マップ上のゴールを指定すると，ゴールの姿勢を選択するダイアログが表示される
- 赤い円内をクリックすると，ゴール姿勢 (マップ座標系での姿勢) を指定できるので，OKする
- マップ上に指定した姿勢でのゴールが出現する (円の中心が位置，直線の伸びている方向がロボットのX方向)

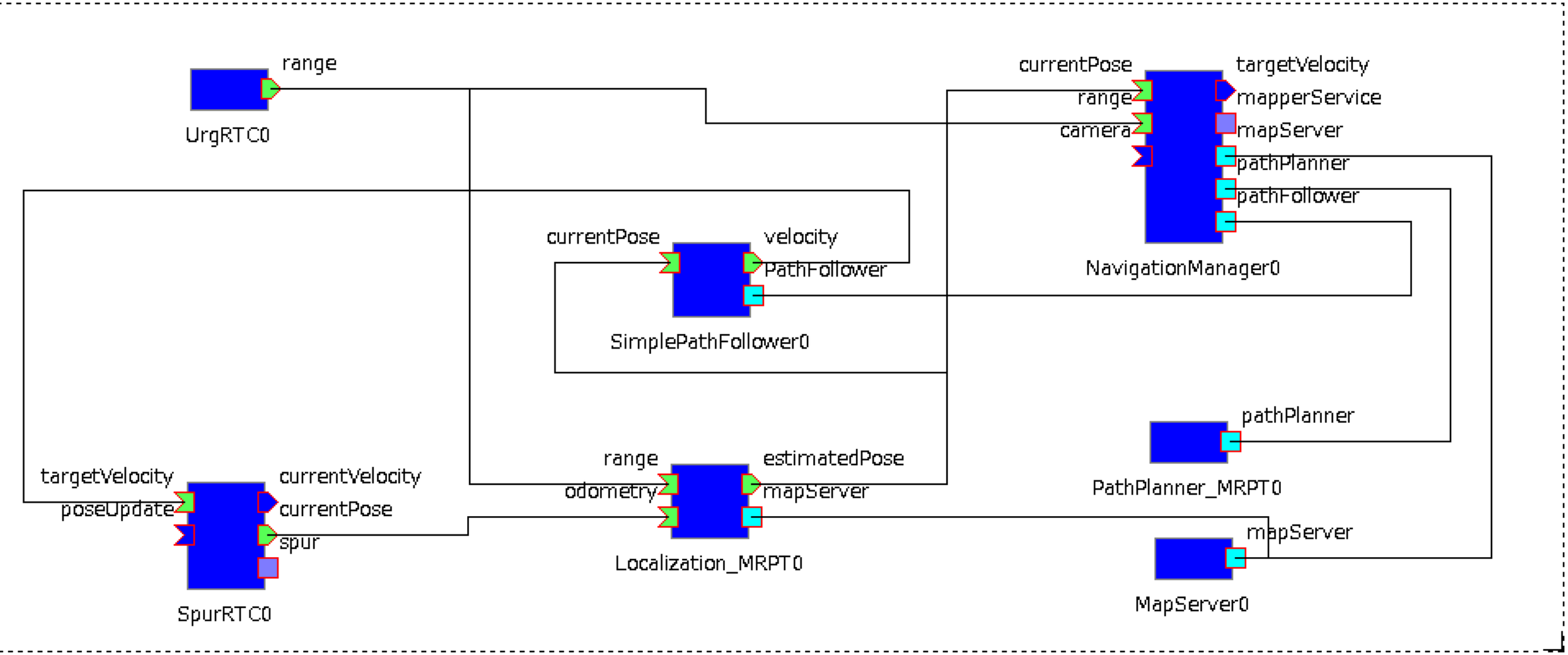


GUIの利用

- Plan Pathボタンでパスプランニング
- Followボタンでロボットが追従開始



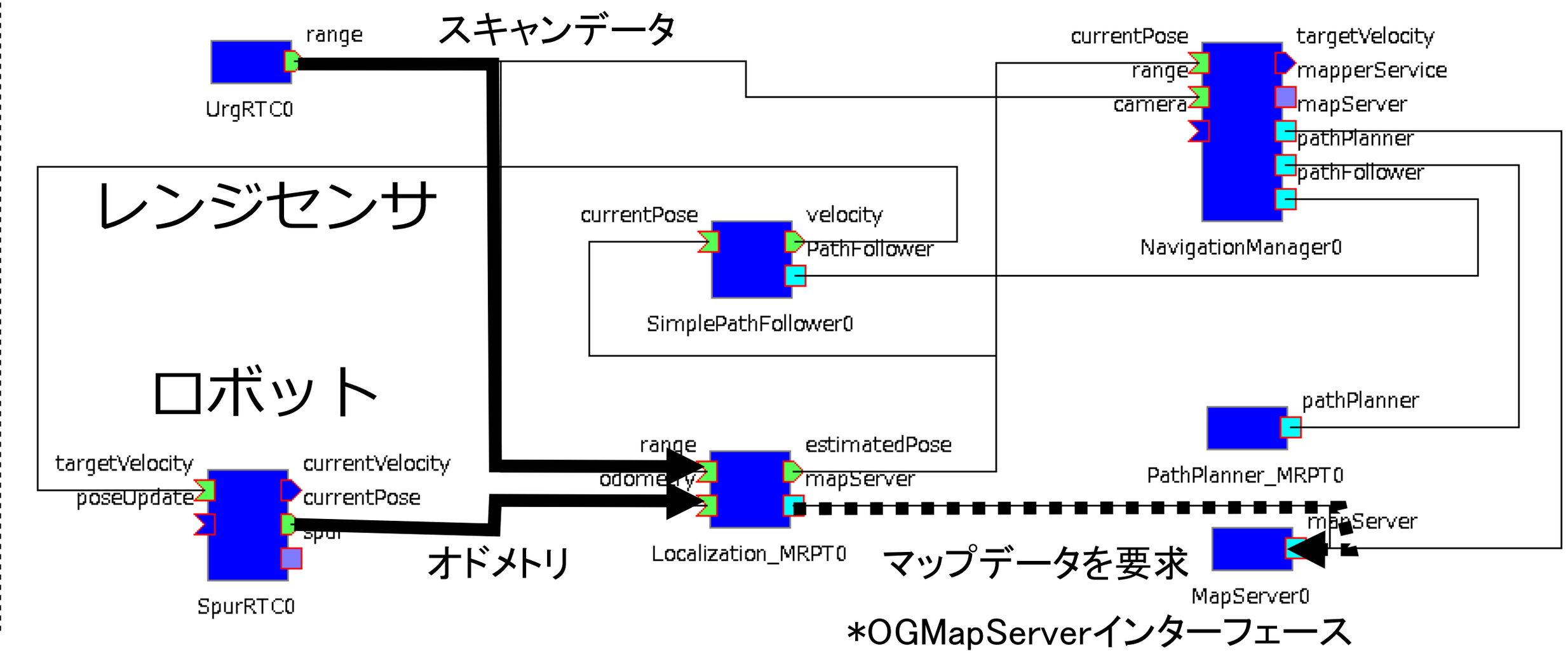
ナビゲーション用フレームワーク (操作とRTシステムとの関係)



菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

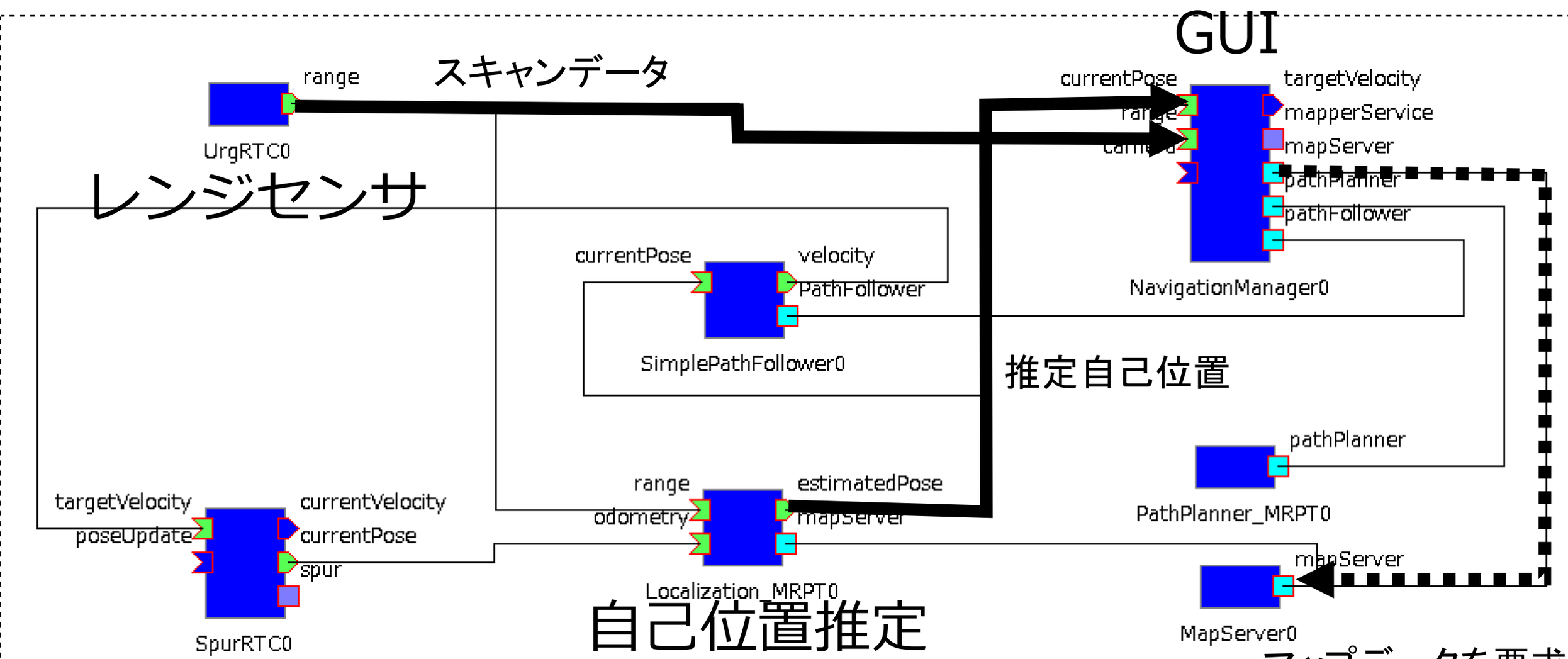
Activate時 (自己位置推定)

- まず，MapServerにマップを要求して取得
- オドメトリとスキャンデータからマップ内での自己位置を推定



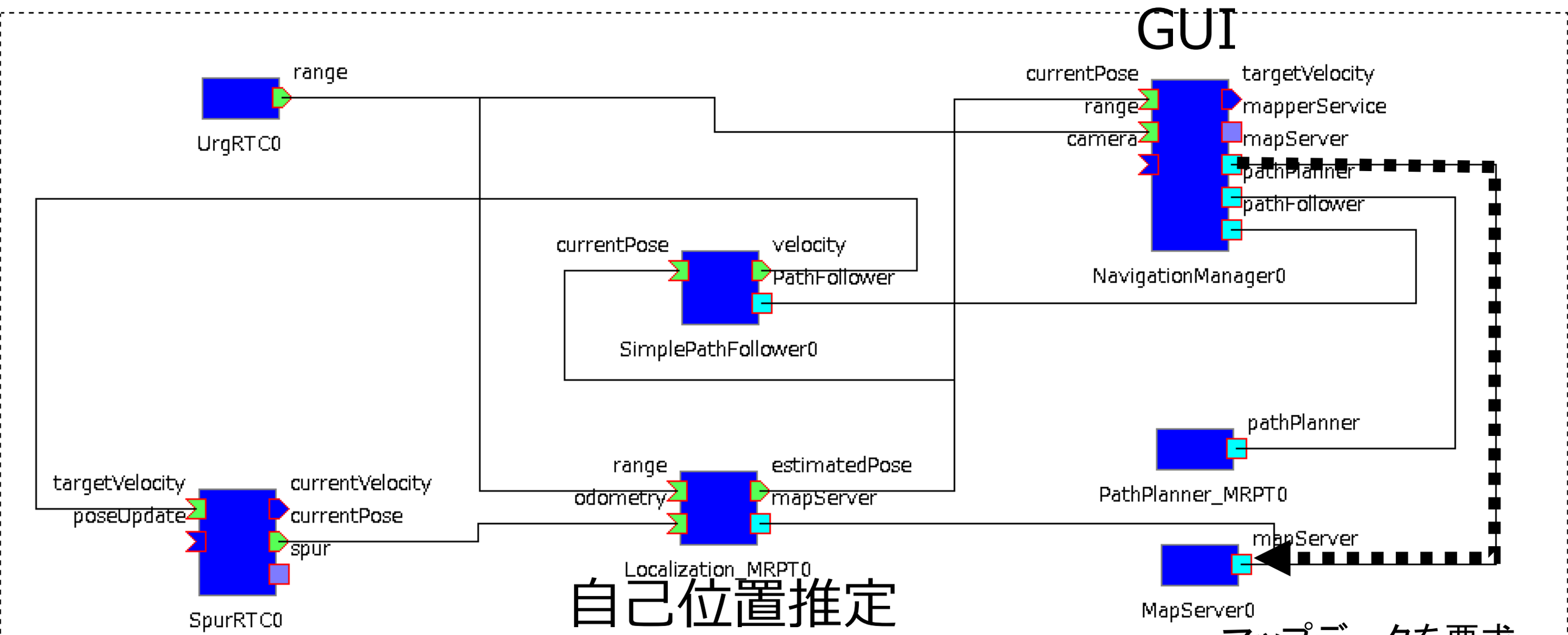
Activate時 (表示)

- GUIはまず，MapServerにマップを要求して取得
- 推定自己位置とスキャンデータをマップに重ねて表示



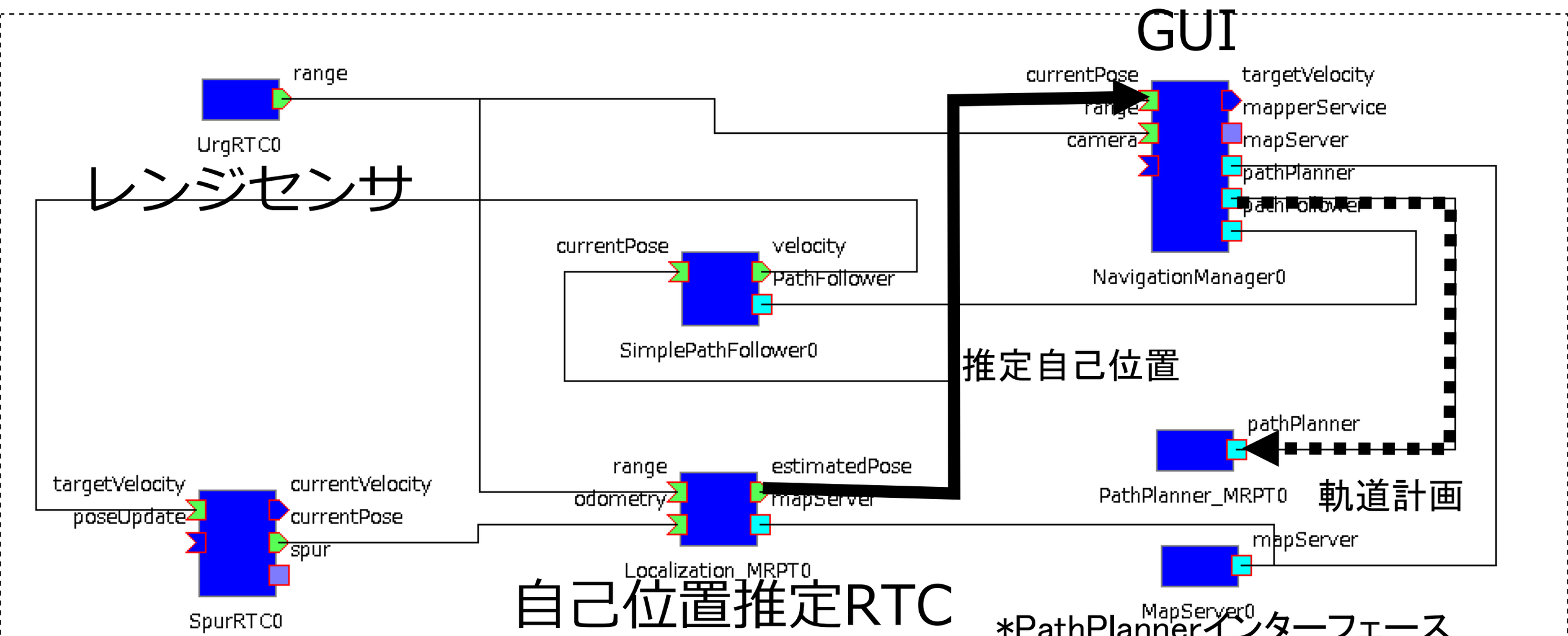
ゴール設定時

- GUI内で処理は完結．クリックした位置を表示したマップ上の位置に変換して記録



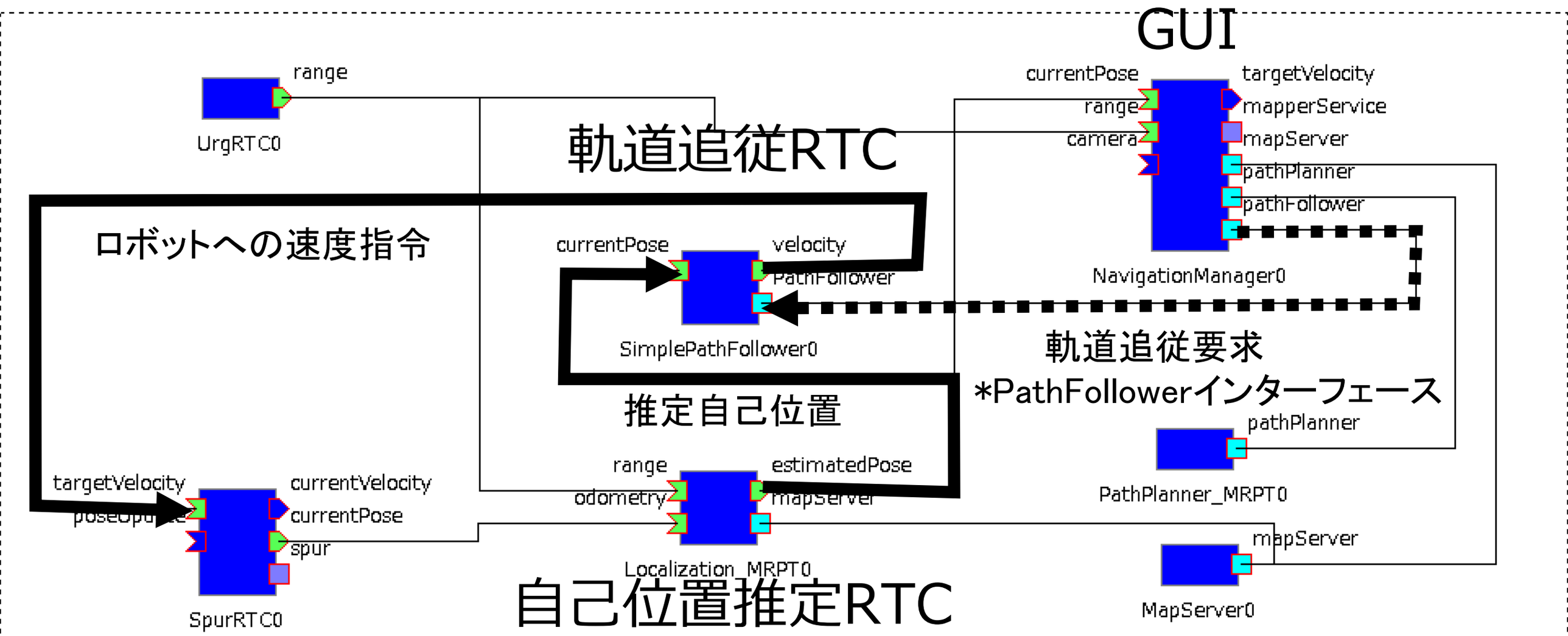
Plan Path時

- ロボットの位置をスタート，指定されたゴール，MapServerから取得したマップ，パスフォローの許容誤差等のパラメータを用いてPathPlannerにプランニングを要求し，Pathを取得・表示

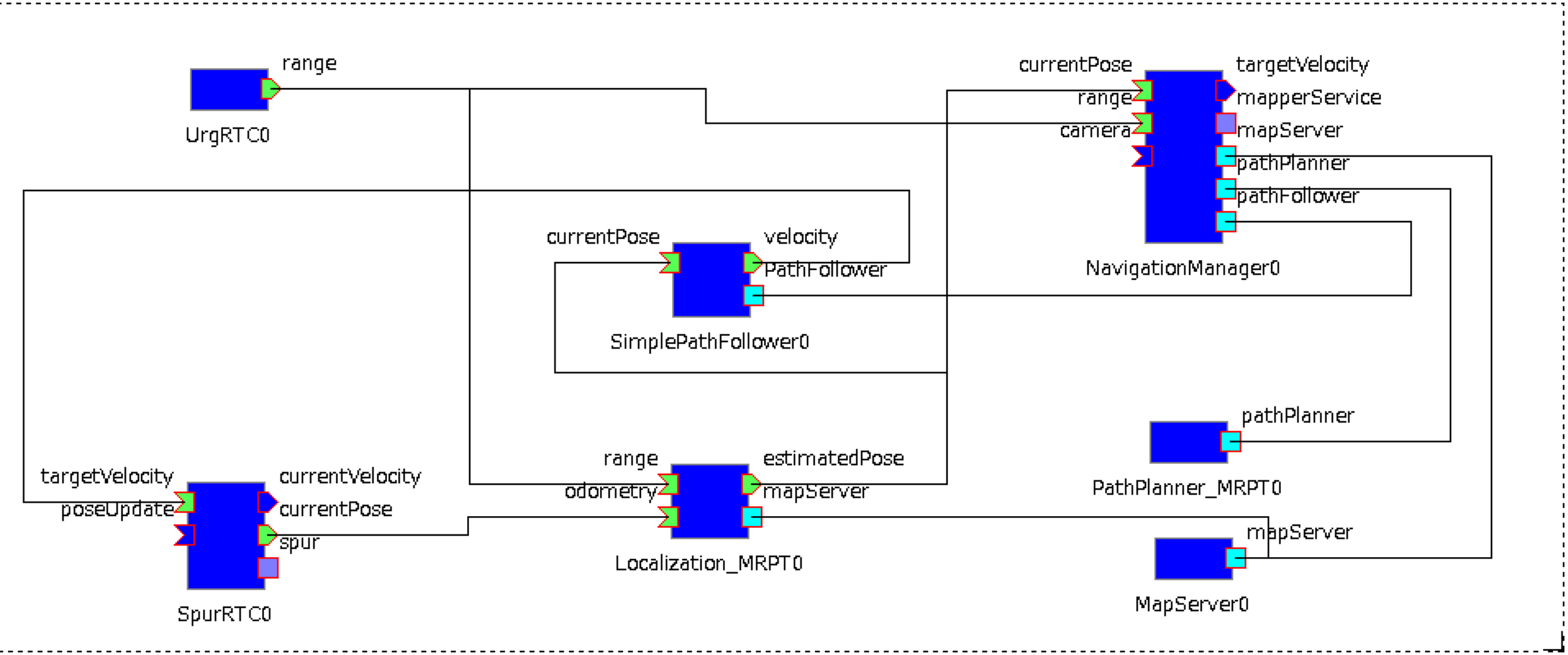


Follow時

- 目標軌道を引数として，軌道追従要求をPathFollowerに要求．成功・失敗を返り値として得る
- ロボットの自己位置と軌道を照らし合わせながら，ロボットの目標速度を送信



ナビゲーション用フレームワーク (各RTCの説明)

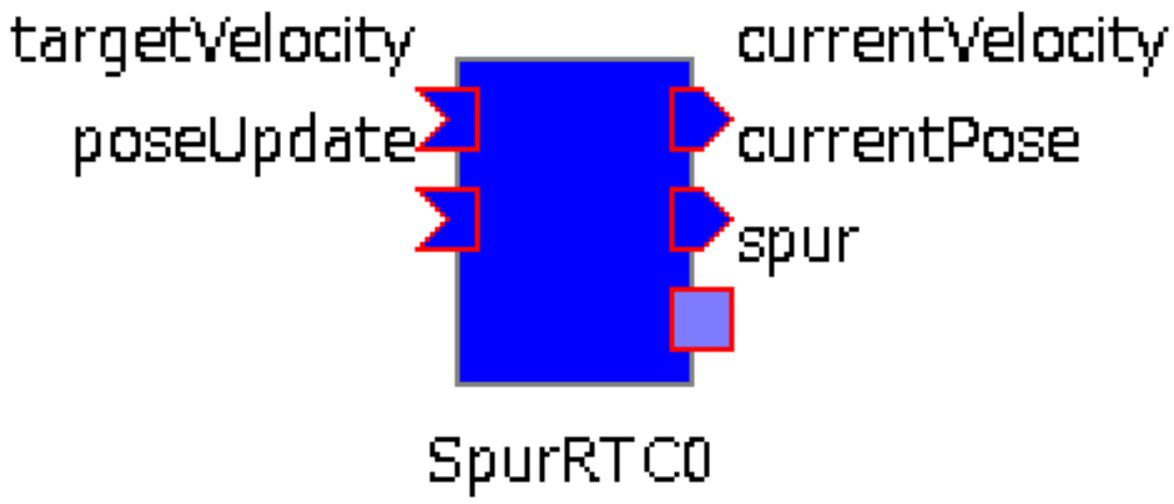


菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

北陽電機 URG

- レーザーセンサ URGのRTC
- 出力ポート
  - range：RangeData
    - レーザースキャナ出力
- 主なコンフィグレーション
  - スキャナのロボット座標系での位置，オフセット
    - geometry_x
    - geometry_y
    - geometry_z
  - COMポート番号
    - port_name

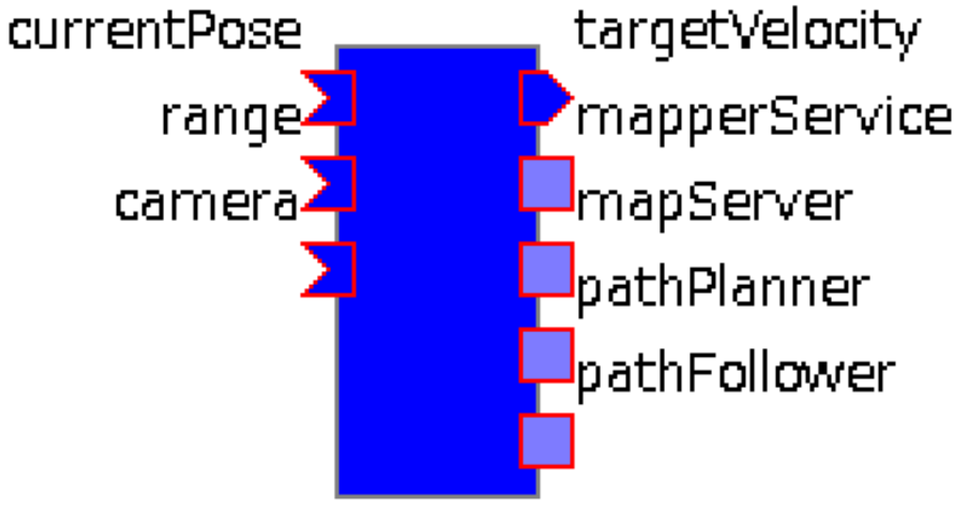
YPSpur対応RTC



- YPSpur対応RTコンポーネント
- 入力ポート
  - targetVelocity：TimedVelocity2D
    - 目標速度
  - poseUpdate：TimedPose2D
    - 現在位置の更新
- 出力ポート
  - currentVelocity：TimedVelocity2D
    - 現在速度
  - currentPose：TimedPose2D
    - 推定自己位置 (オドメトリ)

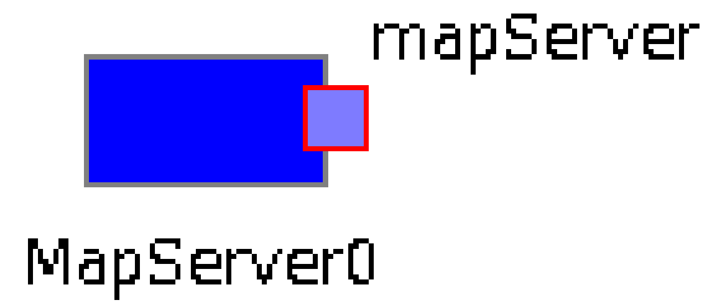
- 主なコンフィグレーション
  - 最大加速度 (activate時に更新)
    - max_acc
    - max_rot_acc
  - 最大速度 (activate時に更新)
    - max_vel
    - max_rot_vel

NavigationManager



- ナビゲーション用GUI
- 入力ポート
  - currentPose：TimedPose2D
    - ロボットの自己位置 (表示用)
  - range：RangeData
    - LiDAR入力 (表示用)
  - camera：CameraImage
    - カメラ画像表示
- 出力ポート
  - targetVelocity：TimedVelocity2D
    - GUIジョイスティックでの操作時に接続
- サービスポート
  - mapServer：RTC::OGMapServer
    - マップ配信モジュールを接続．マップ取得
  - pathPlanner：RTC::PathPlanner
    - 軌道計画モジュールを接続
  - pathFollower：RTC::PathFollower
    - 軌道追従モジュールを接続

MapServer

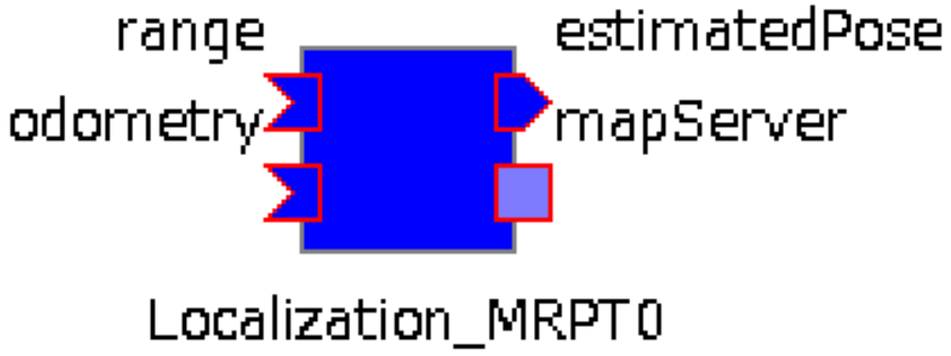


- マップデータを管理・出力
- サービスポート
  - mapServer：MapServer
    - マップデータの出力

- 主なコンフィグレーション
  - filename
    - マップデータのファイル名．Yamlファイルを指定すること．
    - Yamlファイルとpngファイルは同じフォルダに配置すること
    - 実行ディレクトリからの相対パスか，絶対パスで指定．
      - よくわからなければC:¥map¥my_may.yamlなどが無難

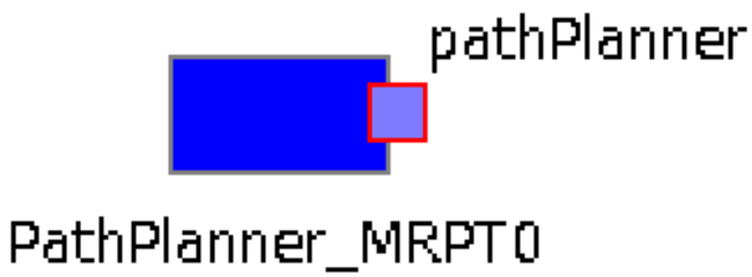
Localization_MRPT

- モンテカルロ法による自己位置推定
- 入力ポート
  - range：RangeData
    - LiDAR入力
  - odometry：TimedPose2D
    - オドメトリ
- 出力ポート
  - estimatedPose：TimedPose2D
    - 推定自己位置
- サービスポート
  - mapServer：MapServer
    - マップデータの取得



- 主なコンフィグレーション
- Dieter Fox，KLD-sampling: Adaptive particle filters. In Advances in Neural Information Processing Systems 14，2002

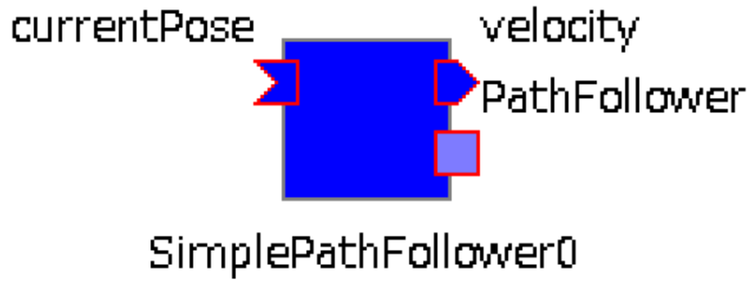
PathPlanner_MRPT



- パスプランニング (MRPT使用)
- サービスポート
  - pathPlanner：PathPlanner
    - パスプランニング

- 主なコンフィグレーション
  - 円柱型ロボット(と仮定した場合)の半径
    - robotRadius
  - 軌道追従の位置・姿勢許容差
    - pathDistanceTolerance
    - pathHeadingTolerance
  - 最終ゴール地点の位置・姿勢許容差
    - goalDistanceTolerance
    - pathHeadingTolerance

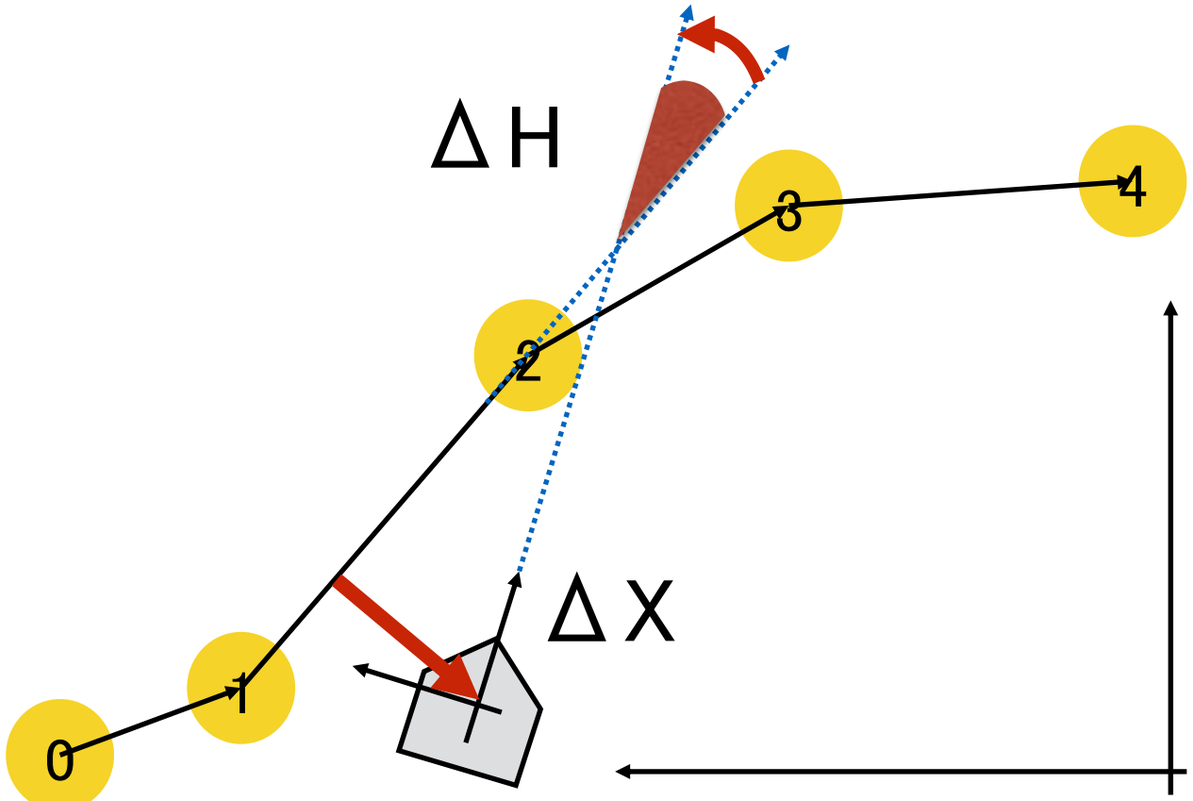
SimplePathFollower



- Path2D型の経路への追従
- 入力ポート
  - currentPose：TimedPose2D
    - 現在位置
- 出力ポート
  - velocity：TimedVelocity2D
    - 目標速度
- サービスポート
  - pathFollower：PathFollower
    - パス追従命令取得

- 主なコンフィグレーション
- distanceToTranslationGain
  - Pathとの位置のズレと並進移動速度のゲイン
- distanceToRotationGain
  - Pathとの位置のズレと回転移動速度のゲイン
- directionToTranslationGain
  - Pathとの向きのズレと並進移動速度のゲイン
- directionToRotationGain
  - Pathとの向きのズレと回転移動速度のゲイン
- approachDistanceGain
  - 最後の終着点との距離と速度のゲイン
- approachDirectionGain
  - 最後の終着点の方向と回転速度のゲイン

SimplePathFollowerアルゴリズム

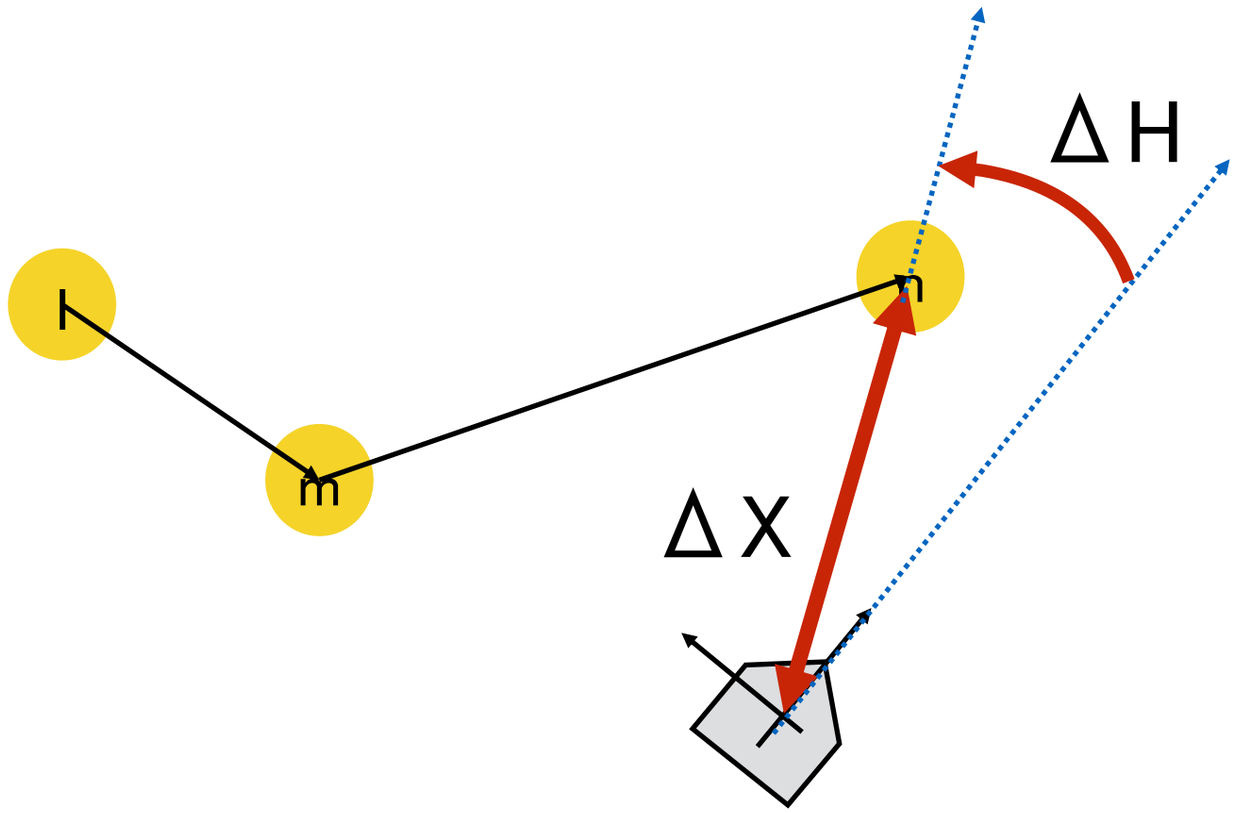


- 最近傍のパスを選ぶ (右図では1-2間)
- 着目したパスとの有効距離をΔX，パスとロボットの向きの差分をΔHとする
- ゲインから速度を計算する
  - Vx = K1 * ΔX + K2 * ΔH
  - Va = K3 * ΔX + K4 * ΔH
- 近傍点のdistanceTolerannceよりもΔXが大きい場合は，OUT_OF_RANGEエラーを返してFollow終了
- ΔHが近傍点のheadingToleranceよりも大きい場合は，Vxをゼロにする

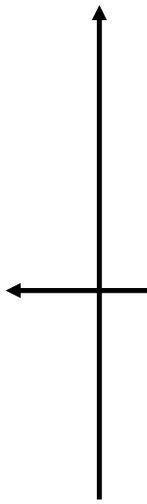
- コンフィグレーションとの対応
  - distanceToTranslationGain = K1
    - Pathとの位置のズレと並進移動速度のゲイン
  - directionToTranslationGain = K2
    - Pathとの向きのズレと並進移動速度のゲイン
  - distanceToRotationGain = K3
    - Pathとの位置のズレと回転移動速度のゲイン
  - directionToRotationGain = K4
    - Pathとの向きのズレと回転移動速度のゲイン

SimplePathFollowerアルゴリズム

- 最近傍パスがゴールを含む場合はゴールアプローチモードになる
- まずゴール点まで移動
  - ゴールとロボットとの距離をΔX
  - ゴール・ロボット間の直線とロボットの向きとの差分をΔH
    - Vx = K5 * ΔX
    - Va = K6 * ΔH
  - ゴール点のdistanceTolerance内で停止
- ゴール点の姿勢までその場で回転し，headingToleranceに収まればゴール到達



- 主なコンフィグレーションとの対応
  - approachDistanceGain = K5
    - 最後の終着点との距離と速度のゲイン
  - approachDirectionGain = K6
    - 最後の終着点の方向と回転速度のゲイン



ナビゲーションフレームワーク応用

- フレームワークのカスタマイズ例
- 独自RTCによるフレームワークの利用

ナビゲーション機能のカスタマイズ

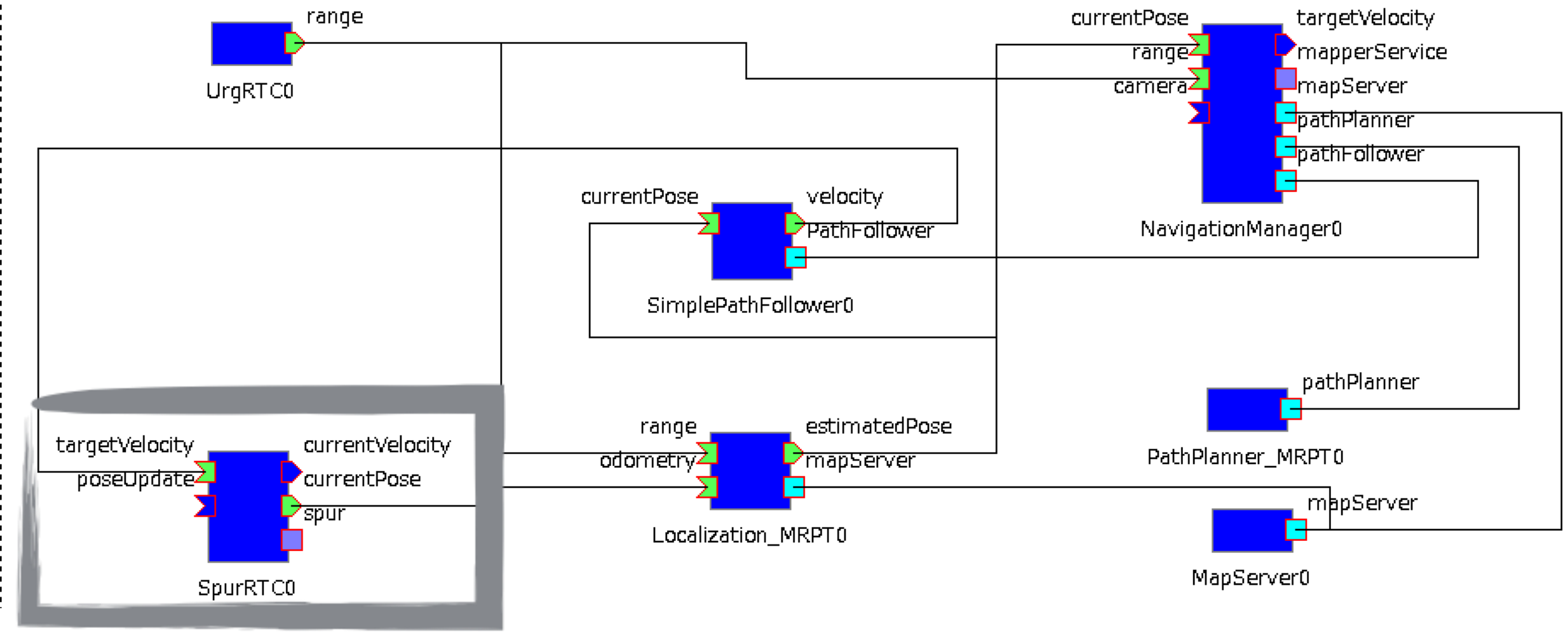
菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

RTCごとの入れ替え

- インターフェースが共通であれば，入れ替えは容易

ロボットRTCの入替え

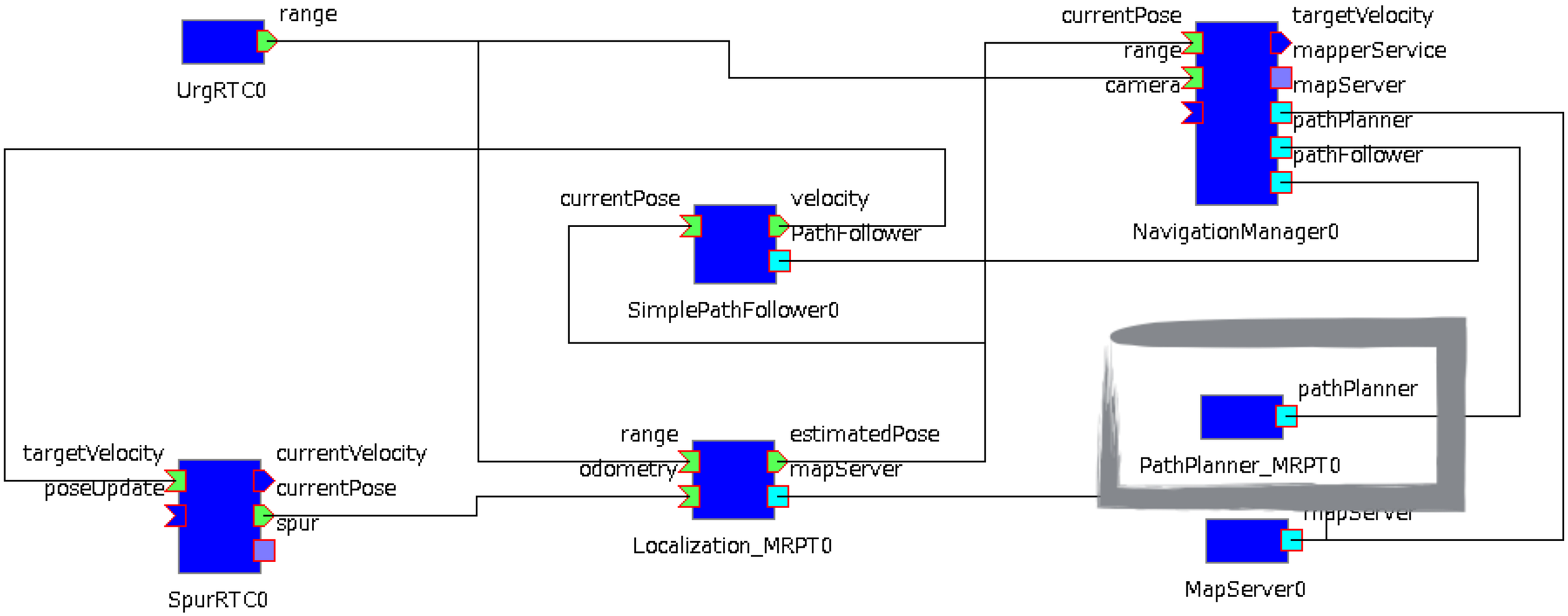
- 独自ロボットへのナビゲーション機能の実装



- TimedVelocity2D入力，TimedPose2D出力

軌道計画RTCの改良

- 軌道計画アルゴリズムを改良
  - RRT-CONNECTなどの実装



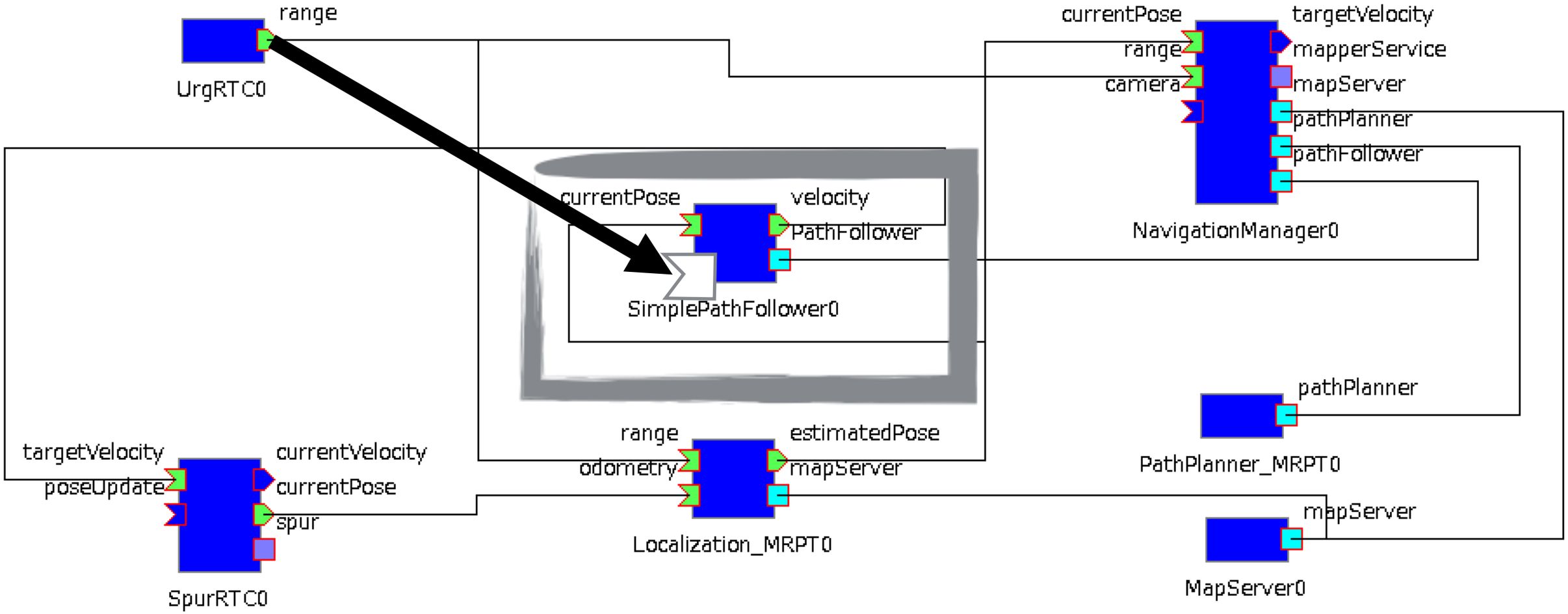
- PathPlannerサービスポート (Provided) を実装

既存RTCの強化

- 外部のRTCと接続するインターフェースが共通であれば，新たな入出力を追加することは可能

軌道追従RTCの改良案

- Urgからのスキャンデータを使って障害物を回避
- ロボットの接触センサ情報を使って衝突時の緊急停止機能を追加



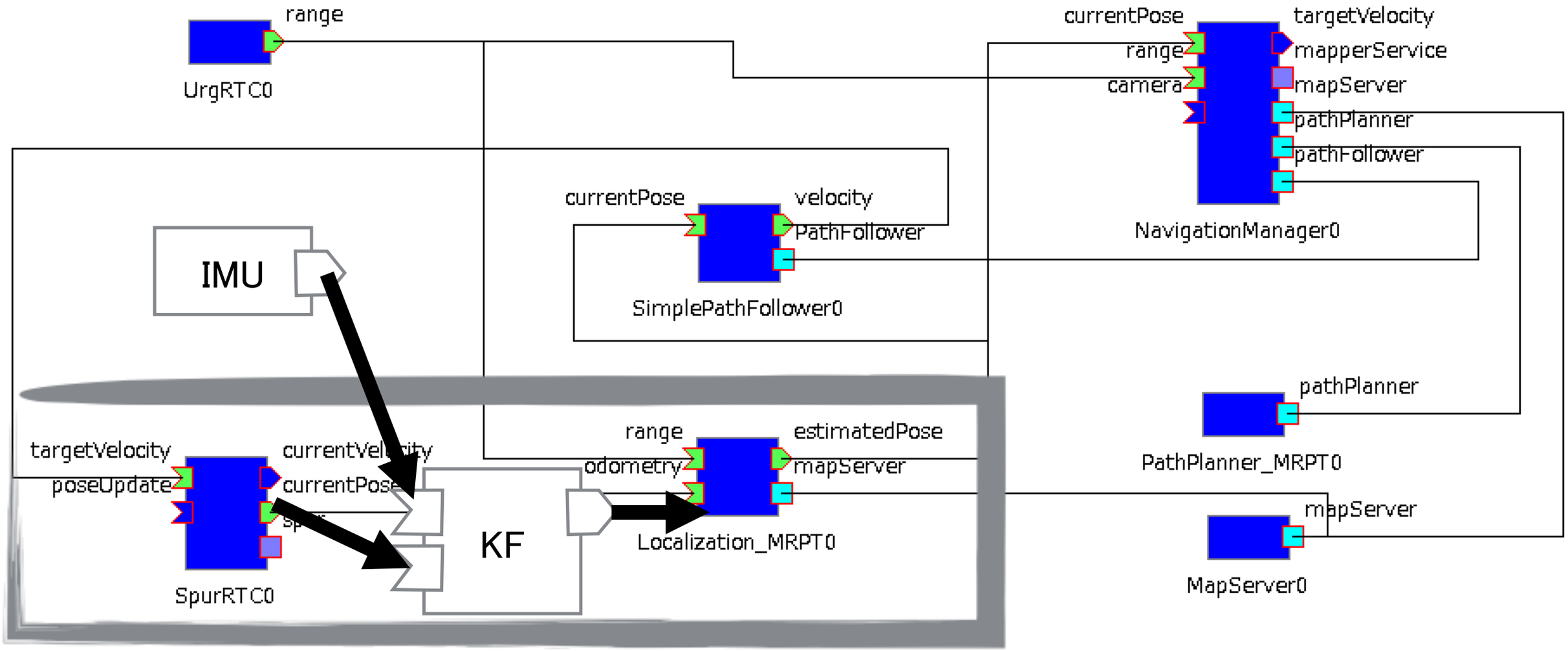
- PathFollowerサービスポート (Provided) を実装
- ロボットの自己位置 (TimedPose2D) を受け取って，速度指令 (TimedVelocity2D) を出力

RTCの追加による フレームワーク機能強化

- 外部のRTCと接続するインターフェースが共通であれば，新たな入出力とRTCを追加することが可能
  - フィルタリング
  - センサーの増設とセンサーフュージョン

自己位置同定の改良

- IMUのRTCを使い，カルマンフィルタを実装して，レンジセンサを使わずに自己位置を出力
- GPSを使って自己位置同定を改良

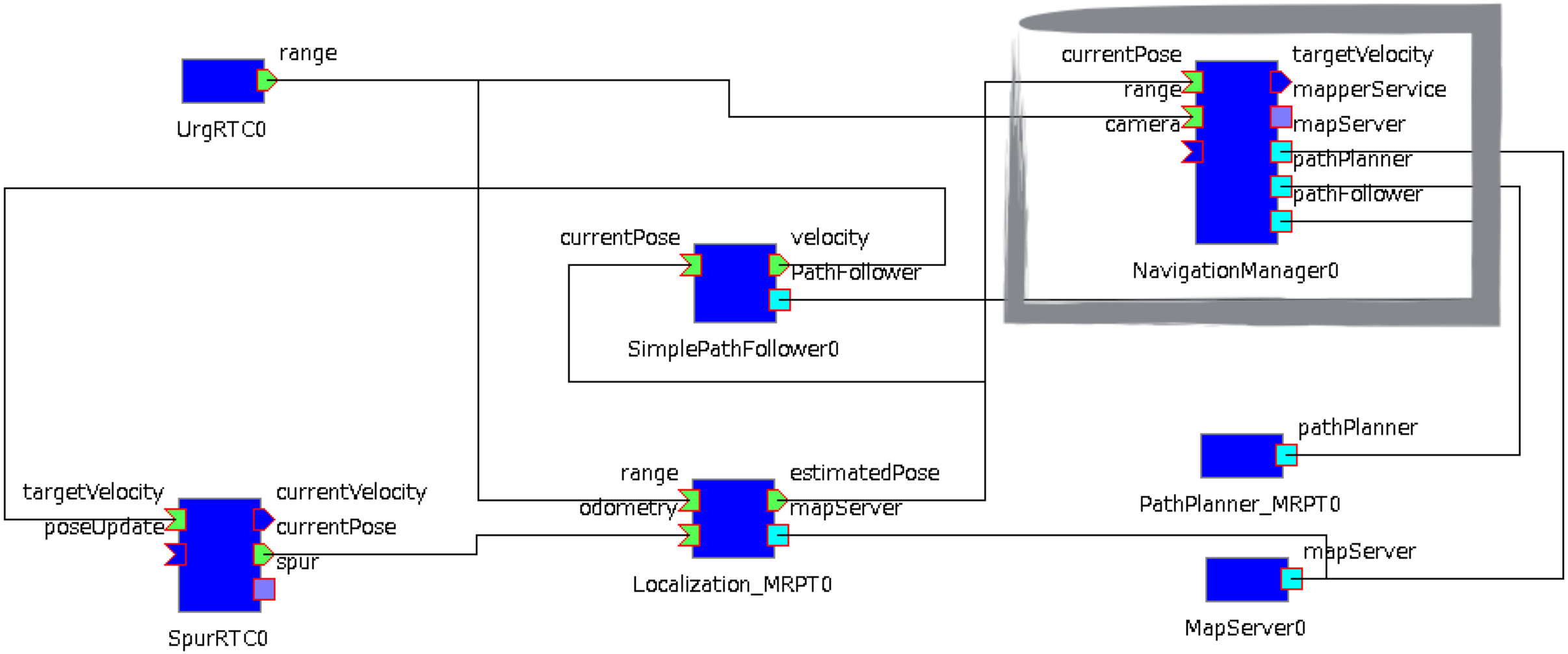


- TimedVelocity2Dを受け取りロボットを動かし，TimedPose2Dでロボットの位置を送信

フレームワーク管理
RTCの追加

菅 佑樹　株式会社SSR × 早大基幹理工表現工学科尾形研究室

一番やりたいのはココですよね？

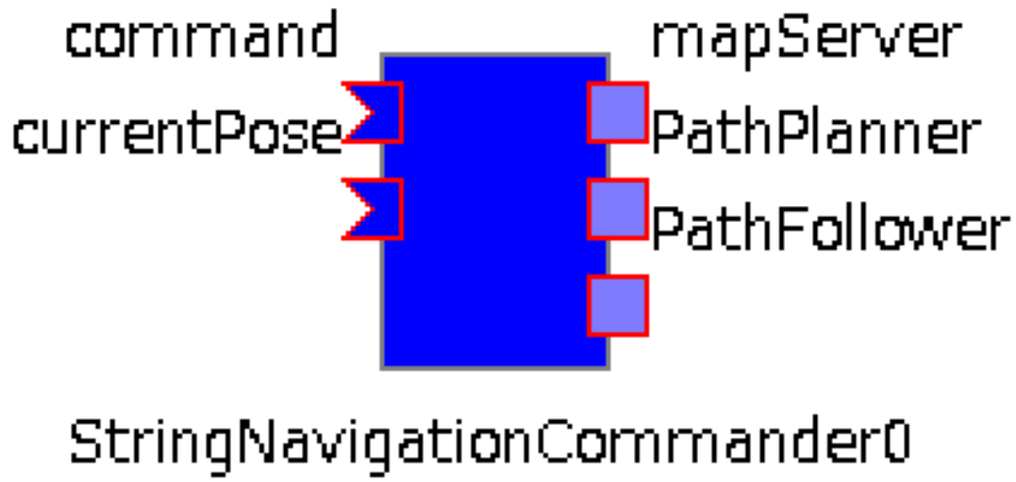


移動ロボットへの指令を変更

- 例えば・・・
  - 音声で「○○に行け」と言ったら，ゴールを自動的に指定して軌道計画，追従を行う
  - 環境センサで状況を判断して自動的に○○を取ってくるロボット
- 上位のサービスの設計をしましょう

StringNavigationCommander

- 文字列命令からゴール位置を設定して，軌道計画・追従を指令するコンポーネント
- 入力ポート
  - command：TimedString
    - GOTO_Xのように指令文字列を受け取る
  - currentPose：TimedPose2D
    - 自己位置を取得 (軌道計画で必要)
- サービスポート
  - mapServer：MapServer (Required)
    - マップを要求・取得
  - pathPlanner：PathPlanner (Required)
    - 軌道計画を要求
  - pathFollower：PathFollower (Required)
    - 軌道追従を要求

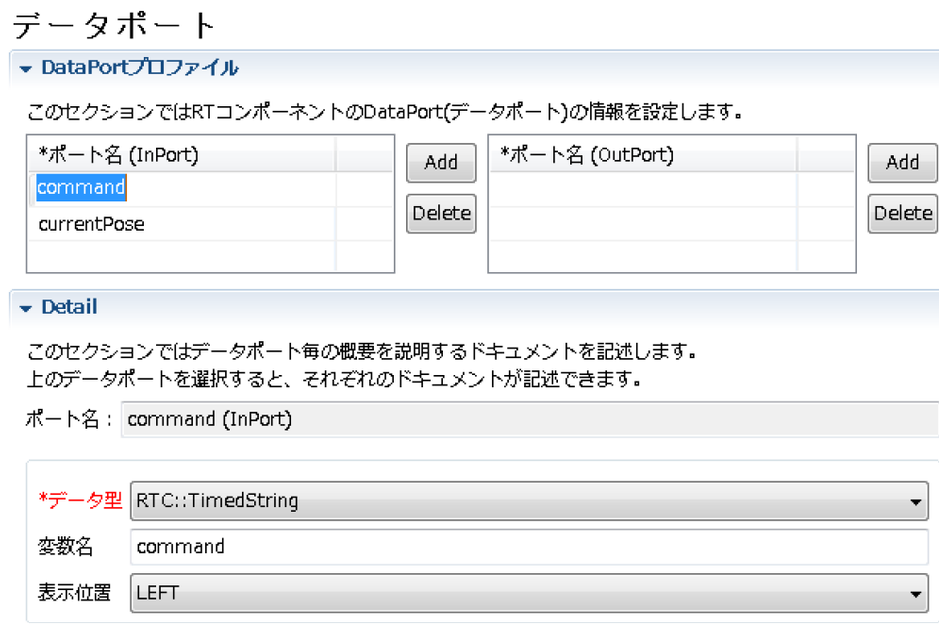


StringNavigationManagerを簡単に作成

- C:¥idlフォルダにIDLファイルをコピー
  - BasicDataType.idl
  - InterfaceDataTypes.idl
  - ExtendedDataTypes.idl
  - MobileRobot.idl
- RTC Builderで新規プロジェクトを作成
  - StringNavigationManagerフォルダのRTC.xmlをインポート

ポートの設定

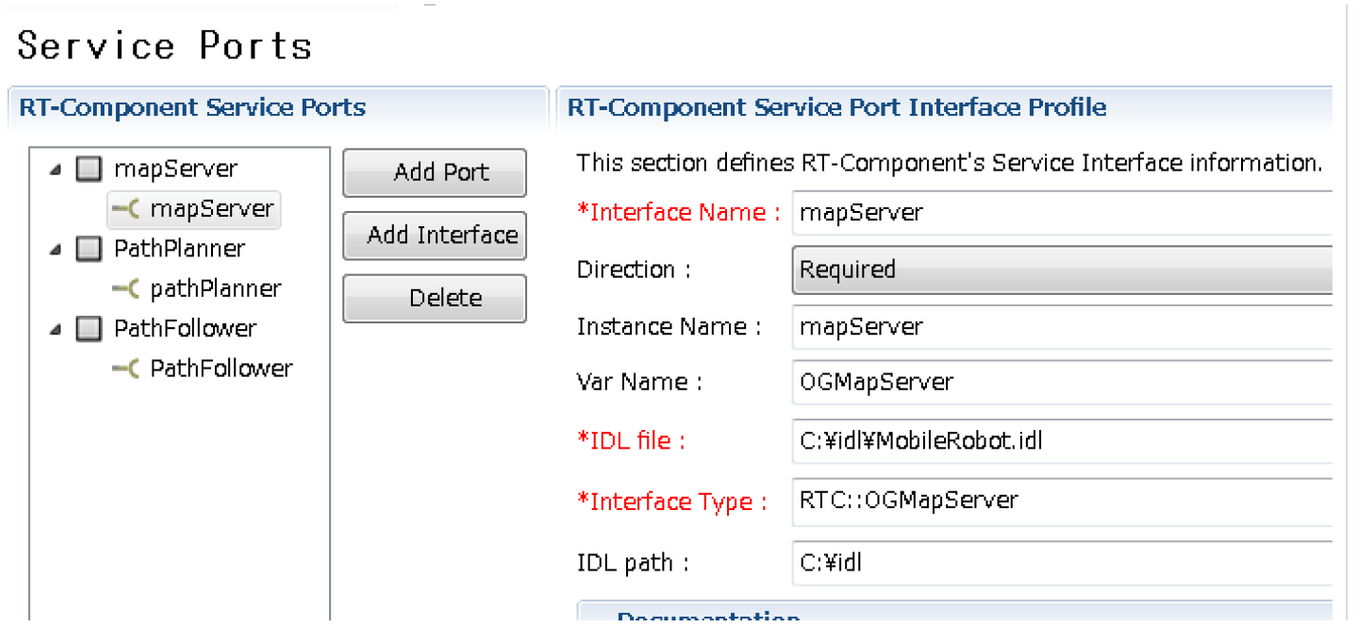
- commandデータ入力ポート



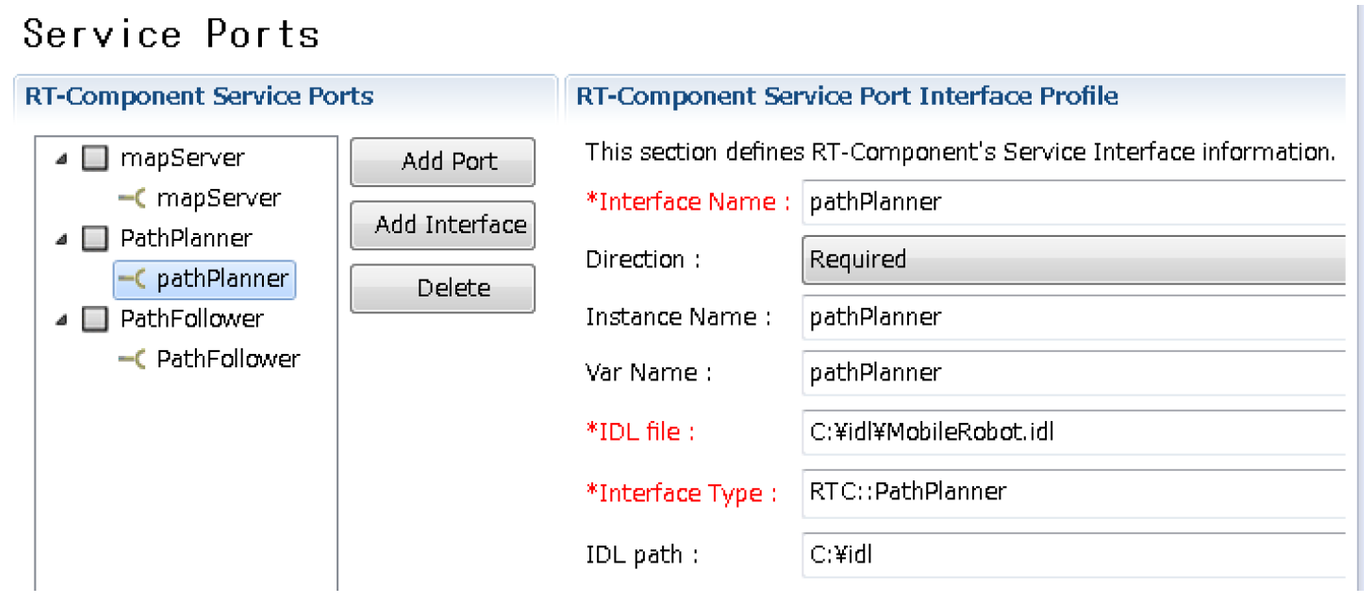
- currentPoseデータ入力ポート

ポートの設定

- mapServerサービスポート
  - Required

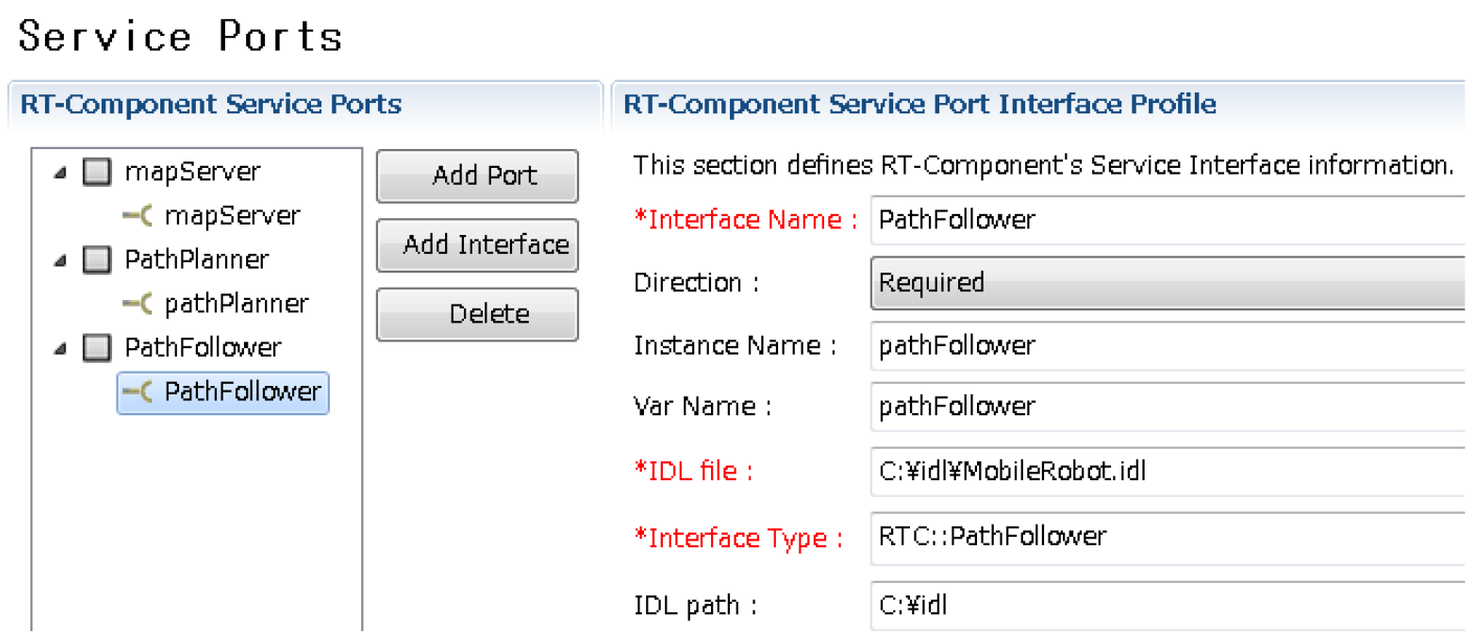


- PathPlannerサービスポート
  - Required



ポートの設定

- pathFollowerサービスポート
  - Required



一旦，通常のビルドを行い接続を確認

- CMakeしてVC++2010でビルド
- ナビゲーション用RTC群を起動して，サービスポートの接続を確認しておく

```
RTC::ReturnCode_t StringNavigationCommander::onExecute(RTC::UniqueId ec_id)
{
    if (自己位置データ受信してたら) {
       自己位置を取得();
    }

    if (文字列コマンド受信してたら) {
       文字列コマンド取得();

       if (コマンド == “GOTO_A”) {
          ゴール = (1.0, 0.0, 1.57);
       } else (コマンド == “GOTO_B”) {
          ゴール = (1.0, 1.0, 3.14);
       } else {
          return RTC::RTC_OK; // 何もせず終了
       }

       マップ = マップ要求();

       軌道 = 軌道計画要求(マップ, ゴール, 自己位置, その他のパラメータ);


       軌道追従要求(軌道);
    }

    return RTC::RTC_OK;
}
```

自己位置取得①

…文字列コマンド取得②

…ゴール位置設定③

…マップ要求④

…軌道計画要求⑤

…軌道追従要求⑥

1. 自己位置データ取得

- 通常のデータポートの方法でデータを取得

```
RTC::ReturnCode_t StringNavigationCommander::onExecute(RTC::UniqueId ec_id)
{
    // 現在のロボットの位置を取得
    if (m_currentPoseIn.isNew()) {
        m_currentPoseIn.read();
    }

...
```

2. 文字列コマンド取得

- 通常のデータポートの方法でデータを取得

```
RTC::ReturnCode_t StringNavigationCommander::onExecute(RTC::UniqueId ec_id)
{

   ...

  // 文字列コマンドを取得
  if (m_commandIn.isNew()) {
      m_commandIn.read(); // m_commandにデータが格納される
      std::string data = m_command.data; // string型に変換
      std::cout << "[RTC::StringCommander] Receive Command (" << data << ")" << std::endl;


...
```

3. ゴール位置設定

取得文字列からゴールを自動設定

```
RTC::ReturnCode_t StringNavigationCommander::onExecute(RTC::UniqueId ec_id)
{

    ...
        // Goalを設定する
        Pose2D goal;
        if (data == "GOTO_A") { // string型は==演算子が使える
            goal.position.x = 1.0;
            goal.position.y = 1.0;
            goal.heading = 1.57;
        } else if (data == "GOTO_B") {
            goal.position.x = 1.0;
            goal.position.y = 0.0;
            goal.heading = 0.0;
        } else { // 知らないコマンドはエラーメッセージを表示
            std::cout << "[RTC::StringCommander] Error. Unknown Command (" << data << ")" << std::endl;
            return RTC::RTC_OK; // onExecuteから通常終了
        }
        // 現在地と目的地が定まる
        std::cout << "[RTC::StringCommander] Current Pose = (" << m_currentPose.data.position.x << ", "
            << m_currentPose.data.position.y << ", " << m_currentPose.data.heading << ")" << std::endl;
        std::cout << "[RTC::StringCommander] Goal Pose    = (" << goal.position.x << ", "
            << goal.position.y << ", " << goal.heading << ")" << std::endl;
```

4. マップ要求

- サービスポートを使ってマップを取得
- out型引数の使い方の例

```
RTC::ReturnCode_t StringNavigationCommander::onExecute(RTC::UniqueId ec_id)
{

    ...

    // マップを要求する
    RTC::OGMap_var outMap;  // マップを格納するための変数

    RTC::RETURN_VALUE retval = this->m_OGMapServer->requestCurrentBuiltMap(outMap);
    if (retval == RTC::RETURN_VALUE::RETVAL_OK) {
        std::cout << "[RTC::StringCommander] Map Request OK." << std::endl;
    } else {
        std::cout << "[RTC::StringCommander] Map Request Unknown Error Code =" << retval << std::endl;
        return RTC::RTC_ERROR;
    }

...
```

5. 軌道計画

サービスポートを使って軌道を取得（out型引数）

```
RTC::ReturnCode_t StringNavigationCommander::onExecute(RTC::UniqueId ec_id)
{

    ...
        // 軌道計画の為のパラメータを調整
        RTC::PathPlanParameter param;
        param.map = outMap; // マップ
        param.currentPose = this->m_currentPose.data; // スタート
        param.targetPose = goal; // ゴール
        param.distanceTolerance = 1.0; // 軌道からの位置のずれの許容差
        param.headingTolerance = 1.0; // 軌道からの角度のずれの許容差
        param.maxSpeed.vx = 5.0; param.maxSpeed.vy = 0.0; param.maxSpeed.va = 1.0; // 最大速度
        param.timeLimit.sec = 9999; param.timeLimit.nsec = 0; // 許容時間
        RTC::Path2D_var outPath; // 軌道を格納するための変数
        retval = m_pathPlanner->planPath(param, outPath);
        if (retval == RTC::RETURN_VALUE::RETVAL_OK) { // 成功
            std::cout << "[RTC::StringCommander] Path Plan OK." << std::endl;
        } else if (retval == RTC::RETURN_VALUE::RETVAL_NOT_FOUND) { // 見つからない（通常終了）
            std::cout << "[RTC::StringCommander] Path Plan No Path Found" << std::endl;
            return RTC::RTC_OK;
        } else {
            std::cout << "[RTC::StringCommander] Path Plan Unknown Error Code =" << retval << std::endl;
            return RTC::RTC_ERROR;
        }
```

6. 軌道追従

- followPathで起動追従．追従終了までブロック

```
RTC::ReturnCode_t StringNavigationCommander::onExecute(RTC::UniqueId ec_id)
{

    …


        // 軌道追従を要求．終了するまでfollowPathはブロックする
        retval = m_pathFollower->followPath(outPath);
        if (retval == RTC::RETURN_VALUE::RETVAL_OK) { // 成功
            std::cout << "[RTC::StringCommander] Follow OK." << std::endl;
        } else { // 失敗も通常終了
            std::cout << "[RTC::StringCommander] Follow Unknown Error Code =" << retval << std::endl;
            return RTC::RTC_OK;
        }


    } // if (m_commandIn.isNew()) に対応
```

```
{
    // 現在のロボットの位置を取得
    if (m_currentPoseIn.isNew()) {
        m_currentPoseIn.read(); // m_currentPoseにデータが格納される
    }

    // 文字列コマンドを取得
    if (m_commandIn.isNew()) {
        m_commandIn.read(); // m_commandにデータが格納される
        std::string data = m_command.data; // string型に変換
        std::cout << "[RTC::StringCommander] Receive Command (" << data << ")" << std::endl;

        // Goalを設定する
        Pose2D goal;
        if (data == "GOTO_A") { // string型は==演算子が使える
            goal.position.x = 1.0;
            goal.position.y = 1.0;
            goal.heading = 1.57;
        } else if (data == "GOTO_B") {
            goal.position.x = 1.0;
            goal.position.y = 0.0;
            goal.heading = 0.0;
        } else { // 知らないコマンドはエラーメッセージを表示
            std::cout << "[RTC::StringCommander] Error. Unknown Command (" << data << ")" << std::endl;
            return RTC::RTC_OK; // onExecuteから通常終了
        }
        // 現在地と目的地が定まる
        std::cout << "[RTC::StringCommander] Current Pose = (" << m_currentPose.data.position.x << ", "
            << m_currentPose.data.position.y << ", " << m_currentPose.data.heading << ")" << std::endl;
        std::cout << "[RTC::StringCommander] Goal Pose    = (" << goal.position.x << ", "
            << goal.position.y << ", " << goal.heading << ")" << std::endl;


        // マップを要求する
        RTC::OGMap_var outMap; // マップを格納するための変数
        RTC::RETURN_VALUE retval = this->m_OGMapServer->requestCurrentBuiltMap(outMap);
        if (retval == RTC::RETURN_VALUE::RETVAL_OK) {
            std::cout << "[RTC::StringCommander] Map Request OK." << std::endl;
        } else {
            std::cout << "[RTC::StringCommander] Map Request Unknown Error Code = " << retval << std::endl;
            return RTC::RTC_ERROR;
        }
    }
    return RTC::RTC_OK;
}
```

```
        // 軌道計画の為のパラメータを調整
        RTC::PathPlanParameter param;
        param.map = outMap; // マップ
        param.currentPose = this->m_currentPose.data; // スタート
        param.targetPose = goal; // ゴール
        param.distanceTolerance = 1.0; // 軌道からの位置のずれの許容差
        param.headingTolerance = 1.0; // 軌道からの角度のずれの許容差
        param.maxSpeed.vx = 5.0; param.maxSpeed.vy = 0.0; param.maxSpeed.va = 1.0; // 最大速度
        param.timeLimit.sec = 9999; param.timeLimit.nsec = 0; // 許容時間
        RTC::Path2D_var outPath; // 軌道を格納するための変数
        retval = m_pathPlanner->planPath(param, outPath);
        if (retval == RTC::RETURN_VALUE::RETVAL_OK) { // 成功
            std::cout << "[RTC::StringCommander] Path Plan OK." << std::endl;
        } else if (retval == RTC::RETURN_VALUE::RETVAL_NOT_FOUND) { // 見つからない（通常終了）
            std::cout << "[RTC::StringCommander] Path Plan No Path Found" << std::endl;
            return RTC::RTC_OK;
        } else {
            std::cout << "[RTC::StringCommander] Path Plan Unknown Error Code =" << retval << std::endl;
            return RTC::RTC_ERROR;
        }

        // 軌道追従を要求．終了するまでfollowPathはブロックする
        retval = m_pathFollower->followPath(outPath);
        if (retval == RTC::RETURN_VALUE::RETVAL_OK) { // 成功
            std::cout << "[RTC::StringCommander] Follow OK." << std::endl;
        } else { // 失敗も通常終了
            std::cout << "[RTC::StringCommander] Follow Unknown Error Code =" << retval << std::endl;
            return RTC::RTC_OK;
        }


    } // if (m_commandIn.isNew()) に対応

    return RTC::RTC_OK;
}
```

さらなる改良案

- ゴール名とゴール位置・姿勢を外部ファイルにして読み込む
  - yamlファイルなどが使い易い

```
GOTO_A : [1.0, 1.0, 0.0],
GOTO_B : [1.0, 0.0, 1.57],
GOTO_C : [1.0, -1.0, 0.0]
```

  - ファイル名をコンフィグレーションにする
- 文字列を音声認識で出力する (OpenHRIを使う)

まとめ

- 台車移動ロボットナビゲーション用フレームワークの紹介
- ナビゲーション用フレームワークのカスタマイズ方法の紹介

- 今後もアップデートを続けていきます
  - ソースコードのダウンロードもこちらから
    - http://ogata-lab.jp/ja/technology_ja.html