# UPnP と ROS を用いたロボットサービスの検出制御システムの提案

江頭　宏亮 *1　　小嶋　奨 *1　　深澤　高雅 *1　　菅谷　みどり *1

## Unified Approach for Service Discovery and Control with UPnP and ROS for Robots

Hiroaki Egashira*1,　Susumu Kojima*1,　Takamasa Fukasawa*1　and Midori Sugaya*1

**Abstract** – **Robots which have multiple services are expected to be used in various places(e.g. office and school). However, since their service discovery and control methods are different due to their hardware components, it is difficult for the users to understand their methods. Furthermore, it causes a decrease of convenience. In this paper, we propose a unified approach for service discovery and control method for robots(SDCR). SDCR achieves a unified service discovery and control method with UPnP for an application working in ROS.**

**Keywords** : **service discovery, service control, UPnP, ROS, robot**

## 1.　はじめに

現在，様々なセンサを搭載した，利用者の用途に合わせプログラム可能である汎用ロボットが注目されている．そのようなロボットは，新たな労働力としてお店やレストランなど様々な場所で活躍しており，活躍の幅を広げている．将来，不足する労働力の確保のため，こうしたロボットの活用は益々進むと考えられる．また，ロボットの提供するサービスは，様々なタスクに対応可能にするため，スマートフォンのように利用者の好きなアプリケーションをダウンロードして利用可能になることが予想される．さらに，家庭や職場などで複数人で共有して利用することも予想される．しかし， ロボットが複数稼働している環境では，利用可能なロボットサービスの検出方法が問題となる．さらに，ロボットごとにサービス制御方法が異なることも問題となる．

汎用ロボットは，用途に合わせてプログラム可能であるため，その汎用ロボットの存在を認識していても，どのようなサービスが利用可能であるのかサービスの利用者が知ることは難しい．例えば，iRobot 社の Roomba [1] であれば，お掃除サービスが利用可能であることはすぐに理解できる．一方で，ソフトバンク社の Pepper [2] のような汎用ロボットの場合，Pepper の存在を認識していても，その Pepper がどのようなサービスが利用可能であるのか知ることはロボットとインスタラクションするまで分からない．そのロボットがどのようなサービスを提供するのかが分かれば，より効率的にサービスを利用することができる．このことから，我々は利用者が，ある場所において，ロボットのサービスを検出する統一的なシステムが必要であると考えた．

二つ目の課題として，利用可能なサービスロボットが特定できた後に，ロボットごとにサービス制御方法が異なる問題がある．現在，ロボットサービスの制御方法はタッチパネルを用いた方法や音声認識を用いた方法など様々なものが存在する．そのため，利用者はそのサービスごとに制御方法を理解し学習し，そのサービスに合わせて対応を変更しなければならない．特に，情報弱者と呼ばれる高齢者などにとっては，サービスごとに新たな制御方法を学習しなければならないことは，大きな負担となる．

このような多様なインターフェイスの問題を解決する方法として，家電を統一的に遠隔から操作することができる iRemocon [3] と呼ばれる製品がある．iRemocon はスマートフォンの統一的なインターフェイスを用いて，赤外線リモコンの代わりの端末として，ネットワークを通じて通信を行い，家電の制御を行う．しかし，iRemocon は事前にユーザが取り扱う家電の機種とリモコンの種類をユーザが自ら設定しなければならない．統一的なインターフェイスによる制御の仕組みを提供することは優れているものの，設定が煩雑であることが問題である．

これまでに，Universal Plug and Play（UPnP）[4], [5] をロボットミドルウェアで使用するための要件についての研究がなされている [6]．その研究は，ロボットでセンサなどのモジュール間通信に UPnP を使用する場合，UPnP にどのような追加要件が必要になるのか述べている．本稿では，ロボットとクライアントアプリケーション間の通信に UPnP を用いるため，

*1: 芝浦工業大学
*1: Shibaura Institute of Technology

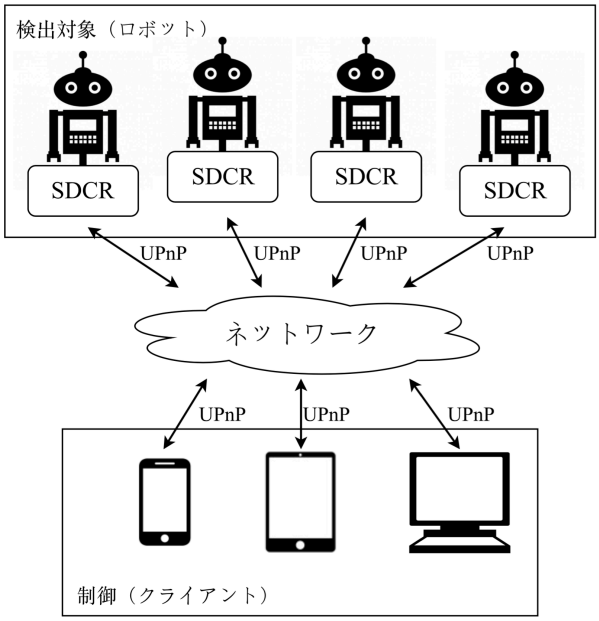


図 1　SDCR の概要
Fig. 1　Overview of SDCR.

UPnP を利用する目的が異なる．

本研究では先に上げた将来的なロボットサービスにおける二つの課題である検出と制御の問題に対して，UPnP と Robot Operating System（ROS）[7] を用いたロボットサービスの統一的な検出制御システム，Service Discovery and Control for Robot(SDCR) を提案する．SDCR は，現在，汎用的に利用されているロボット向けミドルウェアである ROS を用い，UPnP に従ったロボットサービスの統一的な検出と制御を実現する．

本稿は，以下のように構成される．2. 節で SDCR を提案し，3. 節で ROS，4. 節で UPnP の概要を述べる．5. 節で設計と実装について述べる．6. 節で評価，7. 節でまとめを述べる．

## 2.　SDCR の提案

SDCR は，ロボットサービスの統一した検出と制御方法を提供する．SDCR の概要を図 1 に示す．SDCR は ROS と UPnP を基盤としたシステムである．SDCR は複数のロボットを複数人で共有して利用する環境を想定する．SDCR は，ROS を用いて実装され，検出と制御の対象となるロボットサービスも ROS を用いて実装されていることを想定する．そして，クライアントはスマートフォンのアプリケーションやウェブアプリケーションなどのインターフェイスを通して，サービスの検出制御を行う．ロボットとクライアント端末間の通信は，UPnP を用いて行う．

SDCR では，汎用性を考慮して，固有のクライアントアプリケーションを必要としない設計とする．ロボットサービスとクライアントアプリケーション間の通信は，UPnP の仕様に従って行われる．そのため，開発者は UPnP の仕様に従って独自にクライアントアプリケーションを開発可能である．ただし，本研究では評価のためにクライアントアプリケーションの開発も行った．

### 2.1　設計目標

SDCR を設計するにあたり，考慮した点を以下に示す．

1. ネットワークを介して通信を行うことでロボットの位置を意識せずにアクセス可能にする（透過性）
2. 特別な設定を行わずにネットワークに接続するだけで，すぐに SDCR を利用可能にする（効率性）
3. 既存のサービスに特別な変更を加えることなく，SDCR に対応可能にする（再利用性）

（1）について，SDCR は，無線 LAN や有線 LAN などを介してロボットとクライアントアプリケーション間の通信を行う．そのため，ネットワークに接続されていれば，ロボットの位置に関係なくサービス検出制御が可能である．さらに，(2）についても，ロボットやクライアント端末のモデルに関係なく，ネットワークに接続可能な様々なデバイスを SDCR に対応させることが可能である．(3）について，SDCR は，新しいロボットやクライアントアプリケーションを追加しても，ロボットやクライアントアプリケーションに特別な設定を行わずに，すぐに SDCR に対応可能にする．そのため，スケーラビリティが高い．既存のサービスのソースコードに変更を加えることなく，設定ファイルを追加するだけで，SDCR に対応可能にする．そのため，既に稼働しているロボットに容易に SDCR を導入可能である．次節より，SDCR の基盤となる ROS と UPnP について説明する．

## 3.　ROS

ロボットのハードウェアには，ロボットのモデルごとに様々なものが用いられているため，プログラムの再利用が難しい．そこで，プログラムの再利用を支援することを目的として，ロボット向けのオペレーティングシステム，ROS が開発された．ROS のモジュールは，C++や Python などの様々な言語で実装可能であり，ROS の提供する機能を用いることでスムーズに連携可能である．

ROS は，1 つのプロセスのことをノードと呼ぶ．そして，ROS のソフトウェアは複数のノードで構成され，実行時に疎結合することにより，1 つのシステムとして実行可能となる．以下で，ROS の概要について述べる．

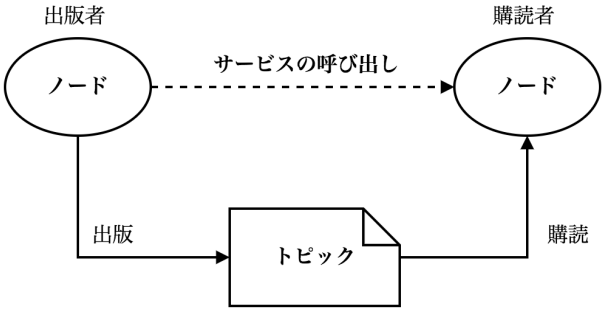


図 2　ROS のノード間通信の概要
Fig. 2　Concept of Communication with Nodes in ROS

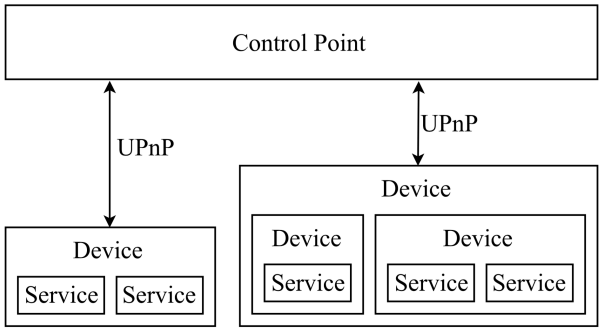


図 3　UPnP 対応機器の構成
Fig. 3　Architecture of UPnP Device.

### 3.1　ノード間の通信

ノード同士は，ROS の通信基盤を用いて通信を行う．ROS のノード間通信の概要を図 2 に示す．通信方法は，トピックを用いた非同期通信とサービスを通じた Remote Procedure call（RPC）型の同期通信の 2 つある．

トピックを用いた通信は，Pub/Sub メッセージングモデルで通信を行う．あるノードはあるトピックにメッセージを配信する．そして，別のノードはそのトピックを購読し，メッセージを受信する．メッセージは，整数型や浮動小数点型，真偽値などの一般的な基本型から成るデータ構造であり，C 言語の構造体のように任意にネストや配列を含むことが可能である．1 つのトピックには複数の配信者や購読者が存在可能であり，配信者も購読者もお互いの存在を意識せずに通信可能である．出版者と購読者の結合度が低いため，スケーラビリティが高い．しかし，非同期通信であり，購読者からレスポンスは得られない．

一方，サービスを通じた通信は，RPC モデルで通信を行う．供給側のノードは，サービス名を指定してサービスを要求し，そのサービスを通じてリクエストを送り，レスポンスを待つ．トピックのメッセージと同様に，リクエストとレスポンスのデータ構造を定義可能である．

### 3.2　パッケージ管理ツール

ROS のノードは，パッケージという単位でグループ化される，パッケージは，ソフトウェアを簡単に共有と配布可能にする．パッケージはノードだけでなく，ライセンス情報や依存関係が記述されたマニフェスト，メッセージのデータ構造を宣言したメーセージタイプ，サービスのリクエストとレスポンスのデータ構造を宣言したサービスタイプなどで構成される．

ROS のソフトウェアは複数のノードで構成されるため，ソフトウェアを起動するには複数のノードを起動しなければならない．複数のノードを起動する煩雑さを解消するために，ROS は roslaunch というツールを提供している．roslaunch とは，ROS 複数のノードを簡単に起動させることを目的としている．roslaunch を利用するには，起動したいノードの情報を XML 形式で記述した roslaunch.xml が必要である．

## 4.　UPnP

UPnP は機器をネットワークに接続するだけで，複雑な設定なしに他の機器との通信を可能にすることを目的としたプロトコルである．UPnP は既に標準化された基本的な通信プロトコルをベースに，取り決めを追加することで，ネットワークに接続した機器の検出や制御を容易に行う．既に広く普及しており，PC やプリンタ，無線機器など，様々な機器を制御するために利用されている．以下で，UPnP の概要について述べる．

### 4.1　UPnP 対応機器の構成

UPnP に対応した機器は，Control Point，Device，Service の 3 つのコンポーネントから構成される．UPnP に対応した機器の構成を図 3 に示す．

Control Point はネットワーク経由で Device を検出し，制御するコンポーネントである．例えば，UPnP を利用して PC からプリンタを制御する場合，PC が Control Point に相当する．Device は Control Point と通信を行い，自身の持つ機能を提供する．この，Device が持つ機能の一つ一つを Service と呼ぶ．先述の例では，プリンタが Device に相当し，印刷する機能が Service に相当する．複数の機器を組み合わせて構成される機器では Device の中に更に複数の Device を持つ構造となる．同様に，Device は複数の Service を持つこともある．

### 4.2　UPnP の通信手順

UPnP は，表 1 に示す 6 つのステップで機器の検出及び制御などを行う．また，UPnP を利用した通信の例を図 4 に示す．

それぞれのステップについて以下で述べる．

Addressing ステップでは，Dynamic Host Config-

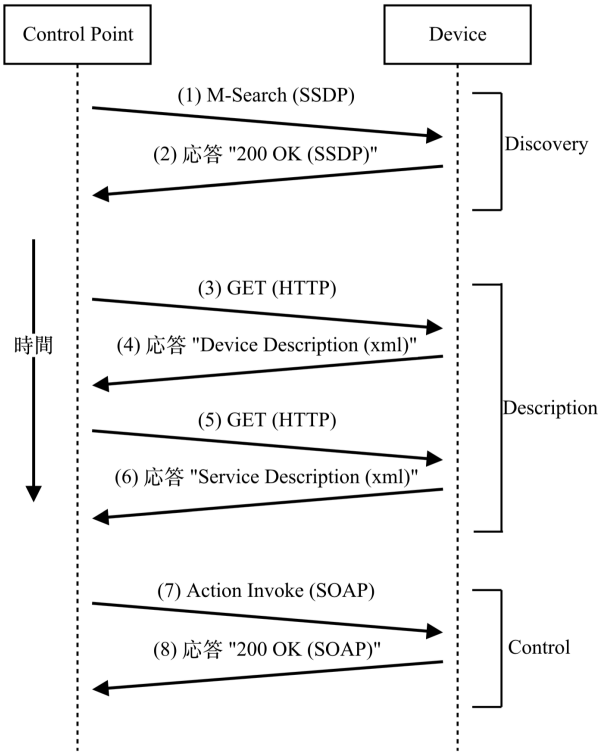


図 4　UPnP を利用した通信例
Fig. 4　Communication Example Using UPnP.

表 1　UPnP の 6 つのステップ
Table 1　six steps of UPnP protocol

| ステップ | 内容 |
|---|---|
| Addressing | Device の IP アドレスの決定 |
| Discovery | Device の検出 |
| Description | Device，Service 情報の取得 |
| Control | Device の制御 |
| Eventing | Device の状態の通知 |
| Presentation | Web コンテンツの提供 |

uration Protocol（DHCP）や，Auto IP と呼ばれる方法を用いて IP アドレスの取得を行う．DHCP は，ネットワークに一時的に接続するコンピュータに対して，自動的に IP アドレスなどの必要な情報を割り当てるためのプロトコルである．Auto IP とは，デバイス自身が IP アドレスをランダムに振り，ネットワーク内に同一の IP アドレスを持つ機器がいないことを確認して IP を決定する方法である．

Discovery ステップでは，ネットワーク機器の発見を行うためのプロトコルである SSDP を利用して Device の検出を行う．UPnP では，マルチキャストグループ 239.255.255.250，ポート番号 1900 番を用いる．Control Point がネットワークに接続されたとき，(1) M-Search メッセージという Device 検出のためのメッセージを UDP マルチキャストで送信する．M-Search メッセージを受け取った Device は，(2) 後述する Device Description を取得するために必要な URL を応答する．また，Device がネットワークに接続されたときは，Advertisement メッセージを UDP マルチキャストで送信する．Advertisement メッセージに記述される情報は，M-Search メッセージに対する応答と同様である．

Description ステップでは，HTTP GET メソッドを利用して，Device の持つサービスなどの詳細な情報が記述された xml を取得する．xml には，Device の情報が記述された Device Description と Service の情報が記述された Service Description の 2 種類が存在する．Control Point は (3) , (5) それぞれ HTTP の GET メソッドを送信することで, (4) , (6) 取得する．Device Description の取得に必要な URL は先述した Discovery ステップで知ることができる．Service Description の取得に必要な URL は Device Description に記述されている．

Control ステップでは，Device の動作を制御する Action Invoke や，Device から変数の値を取得する Query Invoke を行う．Device Description に記述されている controlURL に対して，Control Point は, (7) これらのメッセージを HTTP を使用して SOAP プロトコルで送信する．SOAP とは，XML ベースの RPC プロトコルである．Device は，受信したメッセージに従って制御プログラムを実行する．制御の実行後, (8) SOAP で返り値を含む応答を返す．制御に必要な引数や実行するメソッドの名前などの情報は，Service Description に記述されている．

Eventing ステップでは，Device の持つ変数が変化したとき，Pub/Sub メッセージングモデルで Control Point へ通知を行う．また，Presentation ステップは Device の持つ Web コンテンツの提供をする．

## 5.　設計・実装

SDCR の設計は，検出と制御に ROS と UPnP を基盤として利用する．さらに，本稿では，ROS を UPnP に対応させるために幾つかの設計，実装を行った．ROS は，C++や Python などの様々な言語による開発が可能である．SDCR は，設定ファイルを解析することによりメッセージとサービスのリクエスト，レスポンスの型が決まる．そのため，ランタイムに型の決定可能な Python を用いて実装した．

以下で，SDCR とクライアントアプリケーション間での通信方法について述べる．さらに，サービスの検出及び制御の設計と実装，設定ファイルについて説明する．

### 5.1 SDCR とクライアントアプリケーション間の通信

SDCR とクライアントアプリケーションの間での通信は，UPnP プロトコルに従って行う．SDCR が Device に相当し，クライアントアプリケーションが Control Point に相当する．

検出は UPnP の Discovery ステップと Description ステップを利用する．Discovery ステップを利用して SDCR を利用した機器を発見する．Description ステップにより，Device の詳細な情報，利用できるサービスの情報や制御方法を知る．

制御は UPnP の Control ステップを利用して行う．Action Invoke メッセージに制御に必要な呼び出すメソッドの名称や，引数を格納する．

### 5.2 ロボットとサービスの検出

SDCR でのロボットとサービスの検出では，ROS のソフトウェアはパッケージという単位で形成されるので，パッケージ　サービスと捉えることが可能である．しかし，ROS のインストールと同時に多数のパッケージがインストールされるため，意図しないパッケージをサービスとして検出してしまう．そこで，検出制御の対象となるパッケージとならないパッケージを明確に区別するために，対象となるパッケージには SDCR 用の設定ファイル（sdcr.xml）を設置することにする．sdcr.xml には，サービス情報を XML 形式で記述する．設定ファイルの有無によりサービスの検出を行うので，開発者はソフトウェアのソースコードに変更を加えることなく，ソフトウェアを SDCR に対応させることが可能である．また，SDCR パッケージにも設定ファイル (config.xml) を設置する．そして，SDCR は config.xml を元にロボット情報の取得を行う．

### 5.3 検出手順

SDCR のサービスの検出手順は図 5 のように設計し，実装を行う．まず，SDCR が（1）M-Search メッセージを受信する．すると，SDCR は（2）Device.xml からデバイスの情報を取得する．その後,（3）Device Description 取得用 URL を応答で返す．Device Description 取得用 URL は”http://デバイスの IP アドレス:ポート番号”である．そして，SDCR は（4）先ほど応答で返した Device Description 取得用 URL への GET メソッド要求を受信すると,（5）～（8）SDCR の制御対象となるサービスの検出を行う．SDCR の制御対象となるサービスの検出は，ROS のパッケージ検索の機能を提供する rospack モジュールを用いて実装した．まず,（5）rospack を用いてパッケージ一覧を取得する．そして,（6）取得したパッケージ一覧から SDCR の設定ファイルである sdcr.xml の有無を sys モジュールの isFile() 関数を用いて調べる．そして,（7）,（8）

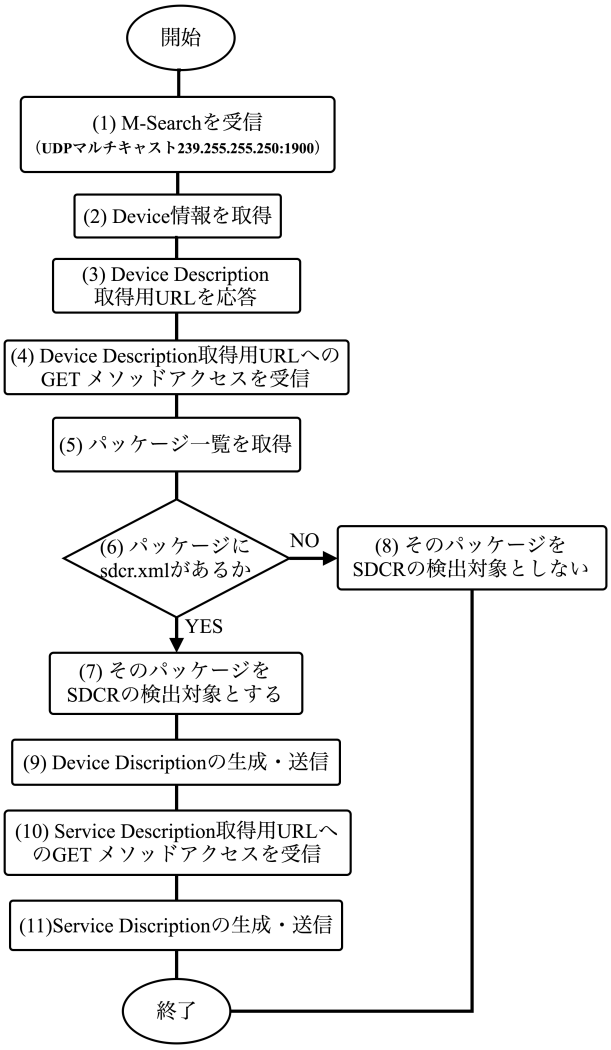


図 5　サービスの検出手順
Fig. 5　Process of Service Discovery.

sdcr.xml の存在するパッケージを SDCR の検出対象のサービスとした．検出対象のサービスが決定すると，検出対象の sdcr.xml の記述から（9）Device Description を生成し，送信する．xml の生成には，XML を扱うためのライブラリである ElementTree [8] を利用する．また，その後,（10）Device Description に記述された Service Description 取得用の URL への GET メソッド要求を受信する．そして，Device Description と同様にして（11）Service Description を生成し，送信する．クライアントアプリケーションは，Device Description と Service Description の記述から利用可能なサービスの情報を知ることができる．

### 5.4 サービスの制御

SDCR でのサービスの制御では，ROS のソフトウェアでは，一般的に roslaunch を用いて起動し，topic と service を用いて制御を行う．したがって，SDCR では，サービスの制御を roslaunch，topic，service を用いて行えるようにする．SDCR が，どのような制御

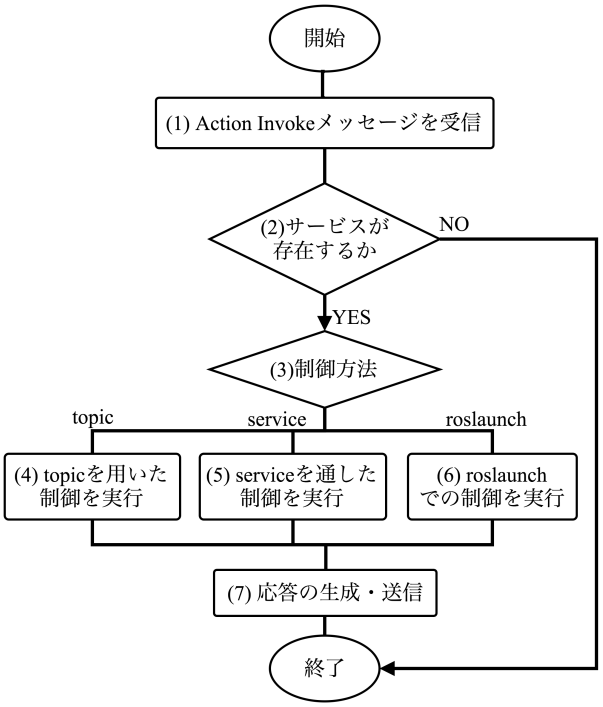


図 6　サービスの制御手順
Fig. 6　Process of Service Control.

方法がサービスにあるのか検出可能にするため，開発者はサービスの制御方法を sdcr.xml に記述する．記述する内容は，サービス名と制御方法，そして，制御方法ごとの情報である．制御方法には,「roslaunch」,「topic」などの制御方法の名前をそのまま記述する，制御方法ごとの情報には，roslaunch の場合，実行に必要なパッケージ名と roslaunch ファイルの名前を記述する．topic の場合，topic の名前，メッセージタイプ，メッセージタイプの型を記述する．service の場合，サービスの名前とサービスのリクエストとレスポンスの型を記述する．

### 5.5　制御手順

SDCR の制御手順は図 6 のように設計し，実装を行う．SDCR は,（1）Action Invoke メッセージを受け取ると，xml の記述から引数や動作の名称を読み出す．そして,（2）サービスが実際に存在するかを確認する．サービスが存在した場合,（3）〜（6）roslaunch, topic, service を用いてサービスの制御を行う．各制御に必要な情報は，各パッケージの sdcr.xml の action タグから取得する．

（4）topic を用いた通信による制御は，rospy パッケージの Publisher クラスの publish(message_instance) メソッドを用いて実装した．メッセージインスタンスの生成は，roslib パッケージの Message クラスの get_message_class(class_name) メソッドを用いて，メッセージのクラス名から生成した．生成したインスタンスのメンバ変数には，genpy パッケージの message クラスの fill_message_args(msg_instance, args) メソッドを用いて，クライアントアプリケーションから受け取った値を代入した．（5）Service を通じた通信は，rospy パッケージの ServiceProxy クラスを用いて生成した Service のインスタンスを用いて実装した．ServiceProxy クラスのコンストラクタは，service 名と service クラスを必要とする．service 名は，sdcrx.xml から取得する．そして．service クラスは，rosservice パッケージの get_service_class_by_name(service_name) メソッドを用いて取得する．service を呼び出すときのリクエストの生成は，topic を用いた通信時と同様に，genpy パッケージの message クラス fill_message_args(msg_instance, args) メソッドを用いて行う．（6）roslaunch によるサービスの起動は，roslaunch パッケージを用いて実装した．そして，制御を終了した後,（7）ElementTree ライブラリを用いて返り値の情報を xml で記述した応答を生成し，送信する．

表 2　作成したクライアントアプリケーションの仕様
Table 2　Specification of The Client Application.

| | |
|---|---|
| OS | iOS 8.0 |
| 開発言語 | Swift 2.2 |
| ライブラリ | CocoaAsyncSocket [9] |
| 実行環境 | iPhone 6s |

## 6.　評価

SDCR の有効性を示すために，iPhone で動作するクライアントアプリケーションを開発した．クライアントアプリケーションの仕様を表 6. に示す．クライアントアプリケーションには，ロボットサービスの検出制御機能を実装した．クライアントアプリケーションのスクリーンショットを図 7 に示す．SDCR は図 7 のメイン画面のように ROS から受けっとったサービス名を表示する．サービスを選択後，図 7 の操作画面のようにサービスを操作するための入力情報を表示する．この時，ROS からデータタイプを受け取り，データタイプに応じた操作方法を提供するので，利用者は統一的なインターフェイスを使用可能である．

### 6.1　検出

SDCR は，無線 LAN などのネットワークを介してロボットと通信を行うことで，ロボットの位置を意識せずにロボットにアクセス可能になった．さらに，UPnP に従ったマルチキャスト通信を用いることで特別な設定を行わずにロボットの検出制御可能になった．また，ロボットに設置した device.xml と sdcr.xml から利用可能なサービスの情報を取得することで，既存のサー


メイン画面　　操作画面

図 7　作成したクライアントアプリケーションのスクリーンショット
Fig. 7　Screenshots of the Developed Client Application.

ビスに特別な変更を加えることなく SDCR に対応可能になった．しかし，sdcr.xml に記述された情報を元にサービスを検出しているので動的にサービスを変更することはできない．

### 6.2　制御

SDCR を用いることで，統一的なインターフェイスでロボットサービスの制御が可能になった．また，開発者は UPnP の仕様に従って自由にクライアントアプリケーションを開発することができるので，利用者は自分で好きなクライアントアプリケーションを開発し，使用することが可能である．

## 7.　まとめ

UPnP や ROS を用いてサービスの検出制御を行うことで SDCR の目的である「ロボットの位置を意識せずにアクセスが可能」や「特別な設定を行わずにネットワークに接続するだけで利用可能」,「既存のサービスに特別な変更が必要ない」を実装することができた．しかし，設定ファイルに記述された情報を元にサービスを検出しているため，動的にサービスを変更できないという問題がある．今後は動的にサービスを変更できるシステム構築が必要である．

## 参考文献

[1] iRobot ロボット掃除機ルンバ 公式サイト, https://www.irobot-jp.com/, (2016/7/23 アクセス)
[2] ロボット | ソフトバンク, http://www.softbank.jp/robot/, (2016/7/23 アクセス)
[3] iRemocon トップページ, http://i-remocon.com/, (2016/7/23 アクセス)
[4] OCF - UPnP, https://openconnectivity.org/upnp/, (2016/7/23 アクセス)
[5] 浜田憲一郎: UPnP 入門, ブイツーソリューション, (2008).
[6] Ahn, S. C.: Requirements to UPnP for robot middleware; 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.4716-4721, (2006)
[7] ROS.org |Powering the world's robots, http://www.ros.org/, (2016/7/23 アクセス)
[8] 19.13. xml.etree.ElementTree - ElementTree XML API - Python 2.7.x ドキュメント, http://docs.python.jp/2/library/xml.etree.elementtree.html, (2016/7/23 アクセス)
[9] Robbie Hanson: GitHub - robbiehanson/CocoaAsyncSocket: Asynchronous socket networking library for Mac and iOS, https://github.com/robbiehanson/CocoaAsyncSocket, (2016/7/23 アクセス)