iHCl 2.0

アダプタ要求仕様書

2017 年 09 月 19 日
rev.1　2017 年 10 月 13 日
rev.2　2017 年 10 月 14 日
rev.3　2017 年 11 月 02 日
rev.4　2017 年 11 月 05 日
rev.5　2017 年 11 月 08 日
rev.6　2017 年 11 月 11 日
rev.7　2017 年 12 月 01 日
rev.8　2017 年 12 月 04 日
rev.9　2017 年 12 月 29 日
rev.10 2018 年 2 月 7 日

ものづくり APS 推進機構（APSOM）

MESX-JP

1. 本仕様書の目的

生産管理システムの要素である既存のソフトウェア製品および新規開発するソフトウェア（MSU, Manufacturing Software Unit）どうしが，階層的な依存関係を持つ状況において，iHCl 言語を使って，クラウド内，クラウド間，クラウドと非クラウド環境で対話するためのアダプタの要求仕様を記述する。

2. 参照文書

- APSOM 00000-2: 2017(CD)，生産管理のための階層間連携–階層間連携言語（iHCl 2.0）によるメッセージとプロトコル
- ISO 16100-3: 2005, Manufacturing software capability profiling for interoperability — Part 3: Interface services, protocols and capability templates[1]

3. iHCl アダプタの機能概要

3.1 対話の構造：Customer-Performer

何らか（出荷，製造，修理など）の注文が，二つの異なる人格，すなわち発注者と受注者の対話で実行される状況がある。発注者と受注者の間でなされる対話は，T. Winograd たちの Customer-Performer モデルで記述できる。生産管理システムにおけるシステム要素間も同様に発注者–受注者の関係にあるとき，依頼側の MSU と受注者の MSU との間で，上記の対話モデルが適用できる。

3.2 通信メッセージ

通信メッセージは，iHCl の規格書にあるとおり，制御メッセージと業務メッセージとからなる。制御メッセージは業務メッセージの通信を制御し，MSU 間の依存性を低め，疎結合を実現する。

業務メッセージは，Customer の役割を果たす MSU（C-Subsystem）と Performer の役割を果たす MSU（P-Subsystem）間で交わされる対話であり，図 1 の状態機械図に基づく。状態遷移のトリガは業務メッセージの一部である遂行動詞（performative）に対応させる。業務メッセージの残りの部分は注文の記述である。

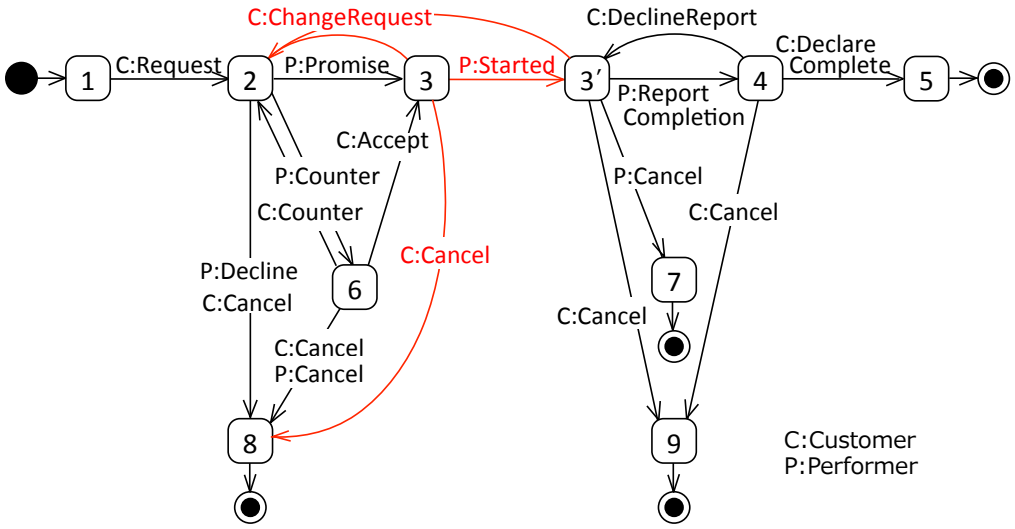


図 1　対話構造と通信メッセージ

[1] JIS B3900-3:2010, 製造用ソフトウェア相互運用のためのケイパビリティプロファイリング–第 3 部:インタフェースサービス，プロトコル及びケイパビリティテンプレート

3.3　iHCl アダプタの役割

iHCl アダプタは，図 2 に示すように，依頼側（C-Subsystem）と受注者（P-Subsystem）の MSU に付属して，対話の相手を特定し，通信メッセージを iHCl にエンコード／デコードして，データをロスすることなく通信する機能を提供する。

制御メッセージのうち，notify は C-Subsystem から P-Subsystem に送りたいメッセージがあることを通知する空メッセージである。P-Subsystem はこのメッセージに基づいて（あるいは基づかないで定期的に）C-Subsystem の pull チャネルに向けて RSVP 制御メッセージを送る。C-Subsystem は RSVP 制御メッセージを受けると，その返信として P-Subsystem 向け（図 1 で P:で始まる）メッセージを組み立てて送る（callback）。

P-Subsystem が C-Subsystem に送信する業務メッセージは，P-Subsystem が C-Subsystem の push チャネルに向けて送信する。

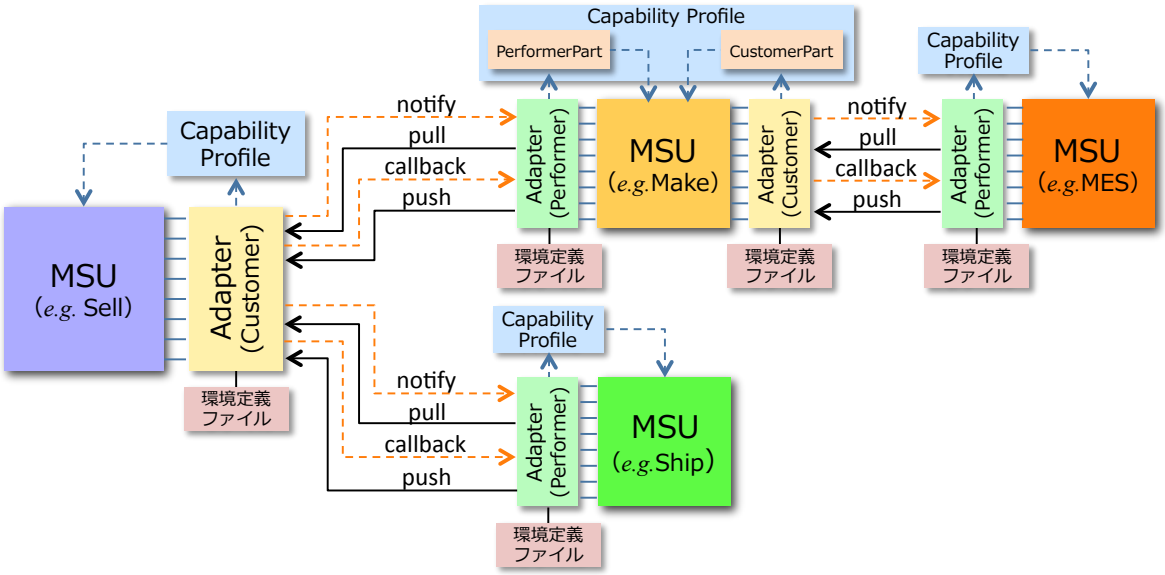


図 2　MSU，アダプタ，通信チャネル

3.4　通信チャネル

通信チャネルは REST-API を用いて実現する。notify と callback は P-Subsystem 側のアダプタに，pull と push は C-Subsystem 側のアダプタに設ける。API の URL は当該 MSU 自身のケイパビリティプロファイルで図 3 のように与える。

通信先のチャネルの情報およびメッセージ情報は，それぞれのアダプタの環境定義ファイルなどに手動で設定する（実装判断）。

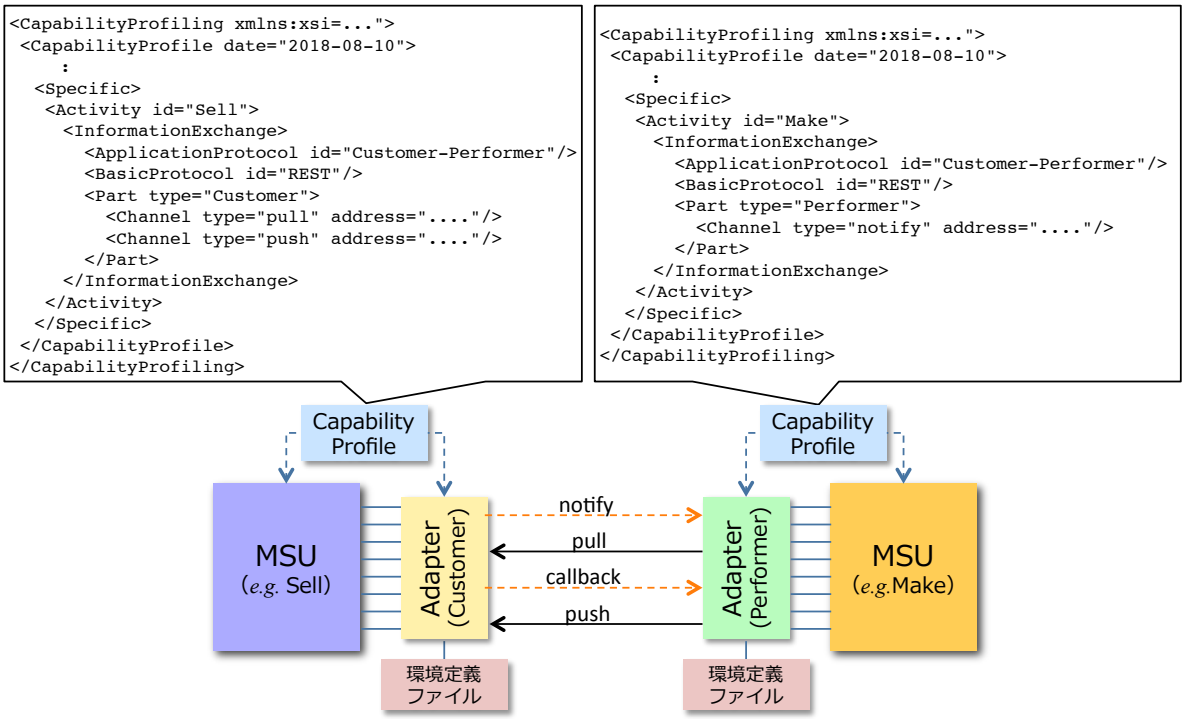


図 3　ケイパビリティプロファイルにおける通信チャネルの定義

3.5　メッセージの変換

アダプタは，MSU から iHCl メッセージのヘッダ情報（メッセージ番号の発番，送信日付の設定，チェックサムの設定など）を挿入し，あるいは除去する。また，メッセージ内容を itemlist（後述）に従って JSON のキーごとに分離し，ケイパビリティプロファイルで指定したデータ型に変換またはその逆を行う。単純にメッセージを仲介するだけであり，プロトコルマトリックスの管理はしない。

メッセージ内容の注文識別をアダプタで発番するか呼び出し側で設定するかを，MSU ごとに選択できるようにする。これはケイパビリティプロファイルではなく，別途設定する設定ファイルの中で指定する。

業種によって異なる業務メッセージの内容や，データ項目のデータ型の情報もケイパビリティプロファイルで指定する。以下はその例である。

```
<Specific>
   <Activity id="Sell" >
      <InformationExchange>
            :
         <Part id="Customer" >
               :
            <Message name="Request">
               <Data name="o-id" type="Identifier"/>       <!-- 注文識別 -->
               <Data name="who" type="Party"/>             <!-- 注文主 -->
               <Data name="what" type="Service"/>          <!-- 注文品 -->
               <Data name="spec" type="JSON"/>             <!-- 注文仕様 -->
               <Data name="qtty" type="Numeric" unit="Kg"/><!-- 数(量) -->
               <Data name="when_to" type="DateTime"/>      <!-- 納期 -->
               <Data name="where_to" type="Address"/>      <!-- 送付先 -->
               <Data name="option" type="JSON"/>           <!-- 追加情報 -->
            </Message>
            <Message name="DeclineReport" />
            <Message name="DeclareComplete" />
         </Part>
      </InformationExchange>
```

```
    </Activity>
  </Specific>
```

4. iHCl アダプタの実装仕様

4.1 低レイヤの通信基盤

HTTP の REST を採用する。加えて, 応答なしのタイムアウトによる通信エラーを検知する。なお, 将来, 低レベルの通信基盤として kafka[2]および ActiveMQ[3]にも対応することを想定しているので，通信基盤とのやりとりを抽象化しておくこと。

4.2 注文番号とメッセージ番号

注文の Request を送信する際，発注者 MSU はアダプタに送信を依頼し，アダプタは当該生産システムにおいてユニークな注文番号を発番し，その注文番号を呼び出し元に返値として渡す[4]。同時に，下行のメッセージのヘッダ情報を組み立てて送信の準備をする。このとき実際の送信は行わない。一時的に蓄積して，受注者 MSU からの RSVP メッセージを待って，その返信としてメッセージを送信する。

メッセージヘッダには，当該生産システムにおいてユニークなでメッセージ番号を発番して，ヘッダに組み込む。このメッセージ番号は対話のコンテクストを維持するために用いられる。

4.3 MSU とのインタフェース

アダプタが MSU に対して提供する API（メソッド）および MSU がアダプタに対して提供するメソッドについて述べる（図 4 参照)。ただし，業務要件に基づいて Customer-Performer の間で交わす対話に必要なメソッドだけを用意する。

4.3.1 Customer アダプタのメソッド

Customer のアダプタは下のメソッドを提供し，Customer の MSU はこれらを必要に応じて使用する。戻り値の ret はメッセージコードで，その値と意味について実装者が定める。

```
Message msg = Request(Order order, DSL itemlist);
int ret = Accept(Message msg);
int ret = Cancel(Message msg);
int ret = Counter(Message msg);
int ret = ChangeRequest(Message msg);
int ret = DeclineReport(Message msg);
int ret = DeclareComplete(Message msg);
```

Request メソッドは，Customer の MSU が Performer に対して注文の送信を依頼する差異に使用する。この引数 order は注文のオブジェクト，itemlist は注文オブジェクトの項目構造を記述した DSL[5]である。DSL の syntax および semantics は実装者からの提案による。Request メソッドの呼び出しにより, order-no が発番され，msg に組み込まれて，その結果が呼び出し元の MSU に返値される。itemlist は呼び出し側，すなわち Customer の MSU が DSL の定義に基づいてセットする。Request 以外のメッセージについても，iHCl の規約に基づいて Customer の MSU で設定する。Customer のアダプタはヘッダ情報を付け加えて iHCl メッセージを組み立てて蓄積しておく。

---

[2] https://kafka.apache.org/
[3] http://activemq.apache.org/amqp.html
[4] 本来，注文番号の発番はアダプタの機能ではない。これはサービス機能である。
[5] データの形式を定義し，変換処理の内容を記述するための特化言語による定義体を想定している。

実際の送信は，Customer のアダプタの Pull チャネルが Performer のアダプタから RSVP 制御メッセージを受け取ったときに行われる。

notify メッセージの送信について Customer の MSU は知る必要はない。Customer の MSU は Request メソッド等を呼ぶだけである。Performer 側の Capability Profile の定義に基づいて，notify メッセージを送信する/しないを決める。Notify による通信の制御を行う場合は，Customer のアダプタが，下行メッセージが 1 件以上蓄積されたことを契機に，Performer のアダプタに Notify 制御メッセージを送信する。Notify を送信する際の条件の設定（n 件以上蓄積されたら notify するなど）は。その具体的な記述形式は実装者の提案による。

4.3.2 Customer MSU のメソッド

Customer の MSU は下のメソッドを提供し，Customer のアダプタはこれらを必要に応じて使用する。戻り値の ret はメッセージコードで，その値と意味について実装者が定める。

```
int ret = Counter(Order order, DSL itemlist);
int ret = Decline(Message msg);
int ret = Cancel(Message msg);
int ret = Promise(Message msg);
int ret = Started(Message msg);
int ret = ReportCompletion(Message msg);
```

Customer のアダプタは, Push チャネル経由で Performer のアダプタから受け取った上行メッセージを，ヘッダの妥当性を確認し，メッセージ本体部分を取り出し，order オブジェクトまたは msg オブジェクトを組み立てた上で MSU のメソッドを呼ぶ。

Customer のアダプタが Counter メッセージを受け取ったとき，逆提案（Counter）を注文形式で見られるように，order オブジェクトを組み立て，order オブジェクトの記載内容を itemlist に定義して，MSU の Counter メソッドを呼ぶ。itemlist の言語については実装者が定める。

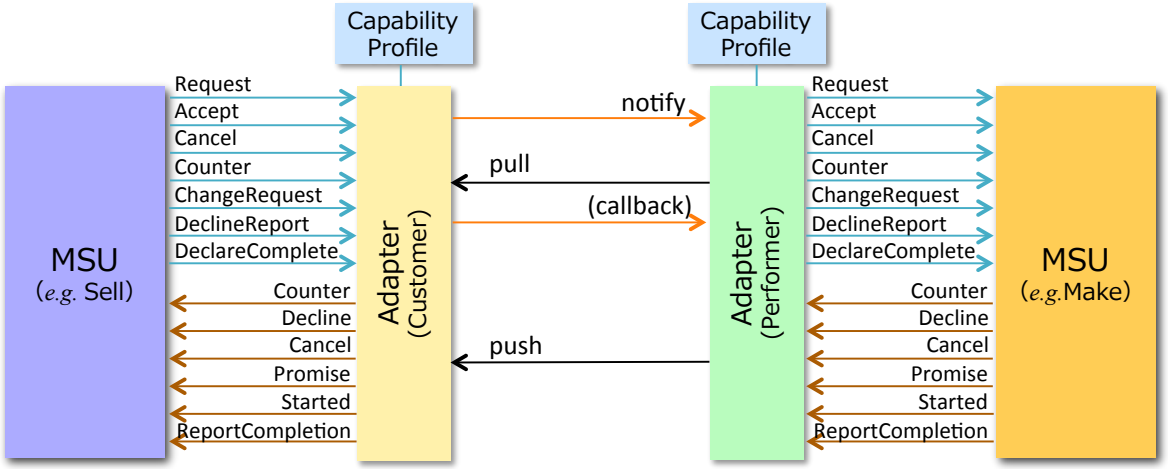


図 4　アダプタおよび MSU が提供するメソッド

4.3.3 Performer MSU のメソッド

Performer の MSU は下のメソッドを提供し，Performer のアダプタはこれらを必要に応じて使用する。戻り値の ret はメッセージコードで，その値と意味について実装者が定める。

```
int ret = Request(Order order, DSL itemlist);
int ret = Counter(Order order, DSL itemlist);
int ret = Accept(Message msg);
```

```
int ret = Cancel(Message msg);
int ret = ChangeRequest(Message msg);
int ret = DeclineReport(Message msg);
int ret = DeclareComplete(Message msg);
```

Performer のアダプタは，Customer のアダプタから Notify 制御メッセージを受け取った時点，または Performer のアダプタに与えられた Polling 間隔時間に基づいて，RSVP 制御メッセージを Customer のアダプタの Pull チャネルに向けて送信する。RSVP 制御メッセージには，Performer のアダプタの callback のアドレスが設定される。callback のアドレスは動的に変更できる。Customer アダプタは，送信すべき下行のメッセージがあれば，RSVP メッセージで示された callback のアドレスに向けてそれらを送信する。

Performer のアダプタは，callback チャネルで受け取った下行のメッセージのヘッダを確認した後に，メッセージ本体部分を取り出し，Performative と itemlist に基づいて引数を組み立てて，上記の Performer MSU のメソッドを呼ぶ。

Performer のアダプタが Request メッセージを受け取ったとき，その内容を注文形式で見られるように，order オブジェクトを組み立て，order オブジェクトの記載内容を itemlist に定義し，MSU の Request メソッドを呼ぶ。itemlist の言語については実装者が定める。Counter メッセージについても同様とする。

4.3.4 Performer アダプタのメソッド

Performer のアダプタは下のメソッドを提供し，Performer の MSU はこれらを必要に応じて使用する。戻り値の ret はメッセージコードで，その値と意味について実装者が定める。

Performer のアダプタは Performer MSU からこれらのメソッドが呼ばれると，直ちに Customer のアダプタの Push チャネルに向けて，該当するメッセージを組み立てて送信する。order オブジェクトの記載内容は itemlist に定義されている。itemlist の言語については実装者が定める。

```
int ret = Counter(Order order, DSL itemlist);
int ret = Decline(Message msg);
int ret = Cancel(Message msg);
int ret = Promise(Message msg, Order order, DSL itemlist);
int ret = Started(Message msg);
int ret = ReportCompletion(Message msg);
```

4.4 アダプタの記述言語および稼働環境

アダプタの記述言語は Java とする。アダプタと MSU との対話は，互いに公開されたメソッドを呼ぶものとする。

OSS のライブラリを使用する場合は，実装前に都度確認すること。ただし，PostgreSQL は確認せずに使用してよい。

アダプタは OS 非依存で稼働すること。ただし，検収テストで使用する OS は，Linux および Windows とし，動作環境は，AWS および IDCF で動作させるほか，MSU によっては，その制約によって自社設置サーバを用いることがある。

4.5 Syntax および Semantics のチェック

アダプタは，ケイパビリティプロファイルおよび itemlist DSL の syntax および semantics の妥当性チェックを行うものとする。違反があった場合は，違反の内容が分かるように表示すること。

4.6　MSU との接続の容易性

アダプタの実装に当たっては，MSU との接続容易性に配慮すること。そのための実装設計の方針やアイデアについて，実装を開始する前にレビューを受けること。

4.7　データロストの対策

メッセージを送信する直前に，メッセージの内容を何らかの形式で永続化（ログ）する。下行メッセージは，いわゆる蓄積交換である。上行メッセージも送信直前にメッセージを永続化（ログ）する。

ログはサイクリックログとし，ログのサイズを外部から与えられるようにする。

4.8　データの抜き出しと注入

アダプタ間のメッセージの抜き出し（escape）および注入（inject）できる拡張点を設ける。これは通信状況の自動テストのためである。本番運用中であっても，テストメッセージを注入し，その処理結果をメッセージにしたものを抜き出して，正常データと照合することで，再帰テストが行える。本番運用には影響を及ぼさない。

escape および inject ポイントごとにテストデータの定義体と起動コマンドを与えることで, アダプタはテストを開始し，すべてのテストデータについての処理が終われば，テストは終了する。結果はログに出力する。機能詳細は別途定める。

4.9　ケイパビリティプロファイルの例

ケイパビリティプロファイルの例を示す。各タグの意味については次項で述べる。実装者は上記の機能を適切に満たすために，この定義方法を合理的に変更してよい。

4.9.1　　ケイパビリティプロファイルのフォーマット

```
<?xml version="1.0" encoding="UTF-8"?>
<CapabilityProfiling xmlns:xsi="http://www.w3.ord/2001/XMLSchema-instance">
   <Template id="iHCl_CapabilityProfilingTemplate.xsd" name="Capability Profiling Template
for iHCl" />
   <Type id="iHCl Capability Profile"/>
   <CapabilityProfile date="20160729">
      <Common>
         <MSU Capability id="....." />
      <Owner name="..." country="..." />
   </Common>
   <Specific>
      <Activity id="業務識別" name="任意文字列">
         <InformationExchange >
            <ApplicationProtocol id="Customer-Performer" />
            <BasicProtocol id="接続方式" />
            <Encoding id="iHCl" version="" />
            <Part type="Customer">
               <Channel type="pull" address="pull URL" >
                  <Method type="http メソッド種類" name="REST API 名">
                     <Param type="パラメタ種類" name="パラメタ名" value="パラメタ値"/>
                     <Param type="パラメタ種類 1" name="パラメタ名 1" value="パラメタ値 1"/>
                     <Auth type="認証種類" login="ログイン認証用 URL" logout="ログアウト用 URL">
                        <Param name="認証情報パラメタ名" value="認証情報パラメタ値"/>
                        <Param name="認証情報パラメタ名 1" value="認証情報パラメタ値 1"/>
                     </Auth>
                  </Method>
               </Channel>
```

```
            :
        </Part>
        <Part type="Performer">
          <Channel type="push" address="push URL" />
            <Method type="http メソッド種類" name="REST API 名">
              <Param type="パラメタ種類" name="パラメタ名" value="パラメタ値"/>
              <Param type="パラメタ種類 1" name="パラメタ名 1" value="パラメタ値 1"/>
              <Auth type="認証種類" login="ログイン認証用 URL" logout="ログアウト用 URL">
                <Param name="認証情報パラメタ名" value="認証情報パラメタ値"/>
                <Param name="認証情報パラメタ名 1" value="認証情報パラメタ値 1"/>
              </Auth>
            </Method>
          </Channel>
            :
        </Part>
      </InformationExchange>
    </Activity>
  </Specific>
```

4.9.2 タグの説明

各タグの意味は次のとおり。

Activity: 対話先のサブシステムごとの記述
  id: MSU 自身の名称で，Sell, Buy/Make, DoMake 等がある。
  name: 名称
InformationExchange: 通信に関する情報定義のヘッダ
ApplicationProtocol: 対話可能な MSU かどうか判定する。
  id: "Customer-Performer"で固定とする。
BasicProtocol:
  id: メッセージ連携で使用するプロトコルの指定。REST 接続の場合は"REST"に固定。以下の説明は REST 前提で述べる。
Encoding: メッセージ形式で省略可
  id: "iHCl"に固定
Part: MSU のロールを定義する
  type: "Customer"または"Performer"で識別する
Channel: チャネルのタイプを指定し，その詳細を定義する
  type: pull/push/notify/callback
  address: 接続先 URL
Method: HTTP メソッド定義
  type: HTTP メソッド種類を定義する。"GET"または"POST"を指定する。
  name: リクエスト送信で使用する REST API を指定する。
Param: HTTP 送信パラメタ
  type: パラメタ種類
  request: リクエストパラメタ
  query: クエリパラメタで，URL の末尾に'?'で連結されて送信される。
  name: パラメタ名称
  value: パラメタ値
Auth: HTTP 認証
  type: Web での認証方式。NONE, BASIC, FORM のいずれか
  login: ログイン認証で使用する API または URL
  logout: ログアウトで使用する API または URL
Param: HTTP 認証パラメタ。'Auth.login'または'Auth.logout'に引き渡されるパラメタ定義
  name: 認証情報パラメタ名称
  value: 認証情報パラメタ値

4.9.3 機能別 Activity の記述例

Sell モジュールの場合。Sell は Customer のケイパビリティのみを持つ。

```
<Activity id="Sell" name="販売せよ">
   <Part type="Customer">
      <InformationExchange >
         <ApplicationProtocol id="Customer-Performer" />
         <BasicProtocol id="REST" />
         <Channel type="pull" address="http://pull のアドレス" >
            <Method type="POST" name="...." />
         </Channel>
         <Channel type="push" address="http://push のアドレス" >
            <Method type="POST" name="...." />
         </Channel>
         <Message name="Counter">
            <Data name="o-id" type="Identifier"/>      <!-- 注文識別 -->
         </Message>
         <Message name="Promise" />
         <Message name="ReportCompletion" />
      </InformationExchange>
   </Part>
</Activity>
```

Buy/Make モジュールの場合。Buy/Make は上位 MSU である Sell に対する Performer のケイパビリティと DoMake などの下位 MSU に対する Customer のケイパビリティを持つ。

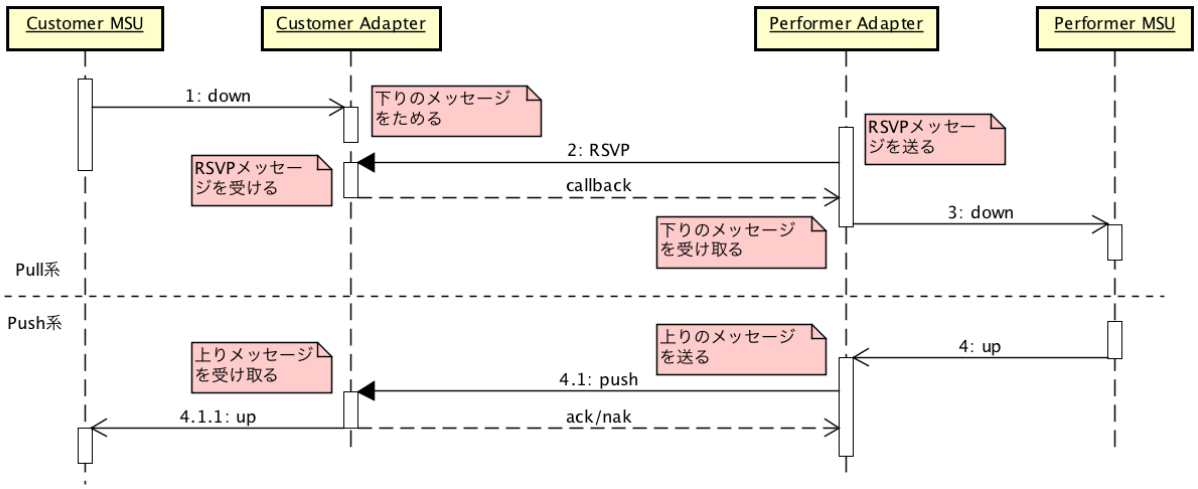
```
<Activity id="Buy/Make" name="購買/製造せよ">
   <Part type="Performer" >
      <InformationExchange>
         <Channel type="notify" >
            <BasicProtocol id="REST" address="http://notify の URL" >
            <Method type="POST" name="...." />
         </Channel>
      </InformationExchange>
   </Part>
   <Part type="Customer" >
      <InformationExchange>
         <Channel type="pull" address="http://pull の URL" >
            <Method type="POST" name="...." />
         </Channel>
         <Channel type="push" address="http://push の URL" >
            <Method type="POST" name="...." />
         </Channel>
      </InformationExchange>
   </Part>
</Activity>
```

DoMake MSU の場合，DoMake は特定の一つ以上の製造工程に対応する。下位の MSU を想定していないので，Performer のケイパビリティのみを持つ。下は notify を用いない場合の例。

```
<Activity id="DoMake" name="製造を実施せよ">
   <Part type="Performer" >
      <InformationExchange>
         <Message name="Request" />
         <Message name="ReportCompletion" />
      </InformationExchange>
   </Part>
</Activity>
```

5. アダプタのユースケース

アダプタのユースケース記述を以下に示す。ユースケース間の関連を下のシーケンス図で示す。



5.1 受注者側のユースケース

5.1.1 RSVP メッセージを送る

アクタ：Performer のアダプタ

概要：Performer のアダプタは，Performer MSU が Ready になったことを確認したのち，Customer のアダプタに対して callback 先を設定した RSVP 制御メッセージを送る。業務メッセージはこの RSVP に対する返事として callback 先に送られる。

事前条件：なし。

事後条件：業務メッセージを受け取る，または何も受け取らない。

基本系列：

①アクタ（Performer アダプタ）とシステム（Performer MSU）との対話が開始された時点で，アクタはこのユースケースを起動する。

②システム（アダプタ自身）は，callback 先も含めて RSVP メッセージを組み立てて，ケイパビリティプロファイルで定義される pull チャネルに当該メッセージを送る。非同期通信なので，返事は待たない。

通信先（Customer アダプタ）は，自分がその時点で蓄積しているメッセージがあれば，それらを回答のメッセージに翻訳し，iHCl メッセージにパッキングして callback 先に送信する。蓄積しているメッセージがないときは，何もしない。

システムは，callback チャネルで下行メッセージを受け取り，これをログに記録した後，ヘッダの妥当性を確認して，ケイパビリティプロファイルの Message タグの data 定義に基づいて翻訳した業務メッセージを（とくに注文データは itemlist に基づいて）組み立てて，Performative に基づいて，Performer MSU にある Request, Accept, Cancel, Counter, Change Request, Decline Report, Declare Complete メソッドを呼ぶ。

システムは，これらメソッドの処理が終了したら一定時間 wait した後，②に行く。wait する前に notify

メッセージが到着していたら直ちに②にいく。

代替系列：

A. 基本系列②の第三段落でメッセージを受け取らなかった場合は，②の第 4 段落に行く。

備考：なし

5.1.2　下りのメッセージを受け取る

アクタ：Performer のアダプタの callback チャネル

概要：Performer のアダプタは，Customer のアダプタが callback チャネルに返信した業務メッセージの提示を受けて，対応する Performer の MSU のメソッドを呼ぶ。

事前条件：Performer のアダプタが業務メッセージを受け取っている。

事後条件：Performer のアダプタが Performer の MSU に業務メッセージを渡している。

基本系列：

①アクタ（Performer アダプタの callback チャネル）が，Performer MSU へ渡す業務メッセージを提示して，このユースケースを起動する。

②システム（Performer アダプタ）は，業務メッセージ内の Performative に応じてそのメッセージをオブジェクト形式に変換し，Performer MSU へ渡す。

代替系列：

A. 基本系列②でオブジェクト形式の変換に失敗した場合は，その旨ログに記録してユースケースを終了する。正式な対応は別途定める。

B. 基本系列②で Performer MSU のメソッドからエラー応答があった場合，その旨ログに記録してユースケースを終了する。正式な対応は別途定める。

備考：

5.1.2　上りのメッセージを送る

アクタ：Performer のアダプタ

概要：Performer のアダプタは，Performer MSU からの送信メッセージの提示を受けて，iHCl 形式に整えた上で，当該業務メッセージを送信する。

事前条件：Performer MSU が Customer に送りたいメッセージがある。

事後条件：Customer に送りたいメッセージが送られた。

基本系列：

①アクタ（Performer アダプタ）が，Customer へ送信する業務メッセージを Performer MSU から受け取って，このユースケースを起動する。

②システム（Performer アダプタ）は，そのメッセージを iHCl 形式に変換して，Customer のケイパビリティプロファイルで定義される push チャネルに当該メッセージを送り，送信されたことを確認する。

代替系列：

A. 基本系列②でメッセージの送信完了（ack 相当）が検知されなかった場合は，その旨ログに記録する。正式な対応は別途定める。

備考：

①メッセージ番号および注文識別の発番処理（システム全体でのユニーク性を保証する），チェックサムの計算，タイムスタンプの設定，メッセージ内容の編集もここで行う。

②Customer アダプタの push チャネル情報は，Customer のケイパビリティプロファイルで定義されているアドレスを，システムの起動の前に，自分の記憶域に記憶しておく。

5.2 発注者側のユースケース

5.2.1 下りのメッセージをためる

アクタ：Customer のアダプタ

概要：Customer のアダプタは，Customer の MSU が Performer に送りたいデータを一旦受け取り，自分のケイパビリティプロファイルの定義，変換ルールを参照して，iHCl 形式に変換したうえで，下りの送信バッファに蓄積しておく。

事前条件：なし

事後条件：業務メッセージが送信バッファに蓄積されている。

基本系列：

①アクタ（Customer の MSU）は，送信したいデータを提示してこのユースケースを起動する。

②システム（アダプタ）は，アクタから送りたいデータを受け取って，Performative ごとに引数を編集して該当するメソッドを呼び出して，iHCl 形式のメッセージ（メッセージコンテンツ）を作り，下りの送信バッファに蓄積しておく。なお，メッセージヘッダは送信時に作成する。

代替系列：

A. 基本系列②で，変換エラーがあった場合，変換エラーである旨をログに記録し，このユースケースを終える。

備考：なし

5.2.2 RSVP メッセージを受ける

アクタ：Customer のアダプタ

概要：Customer のアダプタは，RSVP 制御メッセージを受け取って，その時点で下りの送信バッファに蓄積してあるメッセージにメッセージヘッダを付けて，RSVP メッセージに定義されている callback チャネルに送る。

事前条件：なし

事後条件：下りの送信バッファに蓄積されていた業務メッセージがない。

基本系列：

①アクタ（Customer アダプタ）が Performer からの RSVP メッセージを受け取ったとき，このユースケースを起動する。

②システム（Customer アダプタ）は，その時点で下りの送信バッファに蓄積していた業務メッセージにメッセージヘッダを付けて，RSVP メッセージに定義されている callback に向けて送信する。送信済のメッセージは蓄積リストから削除する。下りの送信バッファに業務メッセージがないときは何もしない。

代替系列：

A. 基本系列②で，callback チャネルからの肯定応答がなかった場合は，通信エラーである旨をログに記録し，下りの送信バッファから当該メッセージを削除しないで①に行く。

備考：

①下りの送信バッファに業務メッセージがないとき，否定応答を返すかどうかは，実装者が定める。

5.2.3　上りのメッセージを受け取る

アクタ：Customer のアダプタ

概要：Customer のアダプタは，Performer のアダプタから受け取った業務メッセージを，メッセージの Performative の値に基づいて Customer の MSU に渡す。

事前条件：なし

事後条件：なし（Customer MSU に業務メッセージが渡された）

基本系列：

①アクタ（Customer のアダプタ）は，受注者アダプタから受け取ったメッセージを提示して，このユースケースを起動する。

②システム（アダプタ）は，受け取った業務メッセージの Performative ごとに用意された変換機能を呼び出して，発注者 MSU が扱いやすい形式に整えて MSU のメソッドを呼ぶ。MSU に送信するメッセージがなくなったらこのユースケースを終える。

代替系列：

A. 基本系列②で，変換エラーがあった場合，変換エラーである旨をログに記録し，このユースケースを終える。

備考：

①Customer MSU が扱いやすい形式とは，ケイパビリティプロファイルに書かれたデータ型に基づく。これ以外の加工は行わない（MSU 自身が行う）。

以上