OpenRTM-aist (Python) - サポート #4121

hrpsys-baseのreadDataPort関数、writeDataPort関数の改善

2017/06/15 11:13 - n-miyamoto

ステータス:	終了	**開始日:**	2017/06/15
優先度:	低め	**期日:**	
担当者:	n-miyamoto	**進捗率:**	100%
カテゴリ:		**予定工数:**	0.00時間
対象バージョン:			

説明

ポートプロファイルにデータを格納する機能の廃止のため、hrpsys-baseのrtm.pyで、データ読み込みのreadDataPort関数、データ書き込みのwriteDataPort関数の処理を変更する。

従来は関数が呼び出されるたびにコネクタの接続、切断を行っていたが、一度生成したコネクタを再度利用するように処理を変更する。

履歴

#1 - 2017/06/15 11:28 - n-miyamoto

- ファイル data.xlsx を追加

- ファイル readDataPortテスト.txt を追加

- ファイル writeDataPortテスト.txt を追加

- ファイル readDataPortテスト.png を追加

- ファイル writeDataPortテスト.png を追加

- ファイル writeDataPortテスト(コネクタ切断あり).png を追加

- ファイル readDataPortテスト(コネクタ切断あり).png を追加

- 進捗率 を 0 から 50 に変更

実装内容

一度生成したコネクタを利用するように変更しました。
また、コネクタが外部から切断された場合は自動的にコネクタを再生成します。

```
connector_list = []

def writeDataPort(port, data, tm=1.0, disconnect=True):
        global connector_list

        connector_name = "writeDataPort"

        prof = None

        for p in connector_list:
                if p["port"]._is_equivalent(port):
                        if port.get_connector_profile(p["prof"].connector_id).name == connector_name:
                                prof = p["prof"]
                        else:
                                connector_list.remove(p)

        if prof is None:
                nv1 = SDOPackage.NameValue("dataport.interface_type", any.to_any("corba_cdr"))
                nv2 = SDOPackage.NameValue("dataport.dataflow_type", any.to_any("Push"))
                nv3 = SDOPackage.NameValue("dataport.subscription_type", any.to_any("flush"))
                con_prof = RTC.ConnectorProfile(connector_name, "", [port], [nv1, nv2, nv3])
                #con_prof = RTC.ConnectorProfile("connector0", "", [port], [nv1, nv2, nv3])
                ret, prof = port.connect(con_prof)

                if ret != RTC.RTC_OK:
                        print("failed to connect")
                        return None
                connector_list.append({"port":port,"prof":prof})
```

```
        for p in prof.properties:
            if p.name == 'dataport.corba_cdr.inport_ior':
                ior = any.from_any(p.value)
                obj = orb.string_to_object(ior)
                inport = obj._narrow(InPortCdr)
                cdr = data2cdr(data)
                if inport.put(cdr) != OpenRTM.PORT_OK:
                    print("failed to put")
                if disconnect:
                    time.sleep(tm)
                    port.disconnect(prof.connector_id)
                    for p in connector_list:
                        if prof.connector_id == p["prof"].connector_id:
                            connector_list.remove(p)
                else:
                    return prof.connector_id
        return None

        global connector_list

        connector_name = "readDataPort"
        prof = None
        for p in connector_list:
            if p["port"]._is_equivalent(port):
                if port.get_connector_profile(p["prof"].connector_id).name == connector_name:
                    prof = p["prof"]
                else:
                    connector_list.remove(p)

        pprof = port.get_port_profile()
        for prop in pprof.properties:
            if prop.name == "dataport.data_type":
                classname = any.from_any(prop.value)

        if prof is None:

            nv1 = SDOPackage.NameValue("dataport.interface_type", any.to_any("corba_cdr"))
            nv2 = SDOPackage.NameValue("dataport.dataflow_type", any.to_any("Pull"))
            nv3 = SDOPackage.NameValue("dataport.subscription_type", any.to_any("flush"))
            con_prof = RTC.ConnectorProfile(connector_name, "", [port], [nv1, nv2, nv3])
            #con_prof = RTC.ConnectorProfile("connector0", "", [port], [nv1, nv2, nv3])
            ret, prof = port.connect(con_prof)

            if ret != RTC.RTC_OK:
                print("failed to connect")
                return None

            connector_list.append({"port":port,"prof":prof})

        for p in prof.properties:
            # print(p.name)
            if p.name == 'dataport.corba_cdr.outport_ior':
                ior = any.from_any(p.value)
                obj = orb.string_to_object(ior)
                outport = obj._narrow(OutPortCdr)
                tm = 0
                while tm < timeout:
                    try:
                        ret, data = outport.get()
                        if ret == OpenRTM.PORT_OK:
                            if disconnect:
                                port.disconnect(prof.connector_id)
                                for p in connector_list:
                                    if prof.connector_id == p["prof"].connector_id:
                                        connector_list.remove(p)

                            tokens = classname.split(':')
                            if len(tokens) == 3:    # for 1.1?
                                classname = tokens[1].replace('/', '.')
                            return cdr2data(data, classname)
                    except:
                        pass
                    time.sleep(0.1)
                    tm = tm + 0.1
```

```
        return None

def delete_all_connector_list():
        global connector_list
        for port in connector_list:
                port["port"].disconnect(port["prof"].connector_id)
        del connector_list[:]
```

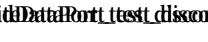
コネクタの名前を一意に決めておいて、get_connector_profilesで生成したIDのコネクタを取得して名前が一致したらコネクタが削除されていないと判定して、そのまま生成済みのコネクタを利用します。※実質的にコネクタプロファイルのnameが空でないかを判定しているだけです。
コネクタが削除されている場合はコネクタの再生成を行います。

動作確認

onExecuteでOutPortのwrite関数を周期的に呼び出すRTC1、InPortのread関数を周期的に呼び出すRTC2を起動して、readDataPort関数、writeDataPort関数に要する時間を計測しました。
これで7時間ほど放置した結果は以下のようになりました。


データ書き込み、読み込みに要した時間のほとんどは0.004秒以内ぐらいで処理できているのですが、1時間に1回程度処理に大きな時間がかかる現象が発生しています。原因は特定できていません。

コネクタが外部から切断される場合もあると考えて、コネクタを周期的に切断して動作を確認してみました。結果は以下のようにコネクタ切断時は時間がかかるのですが、それ以外は0.004秒以内ぐらいで処理できています。



#2 - 2017/06/20 20:24 - n-miyamoto

- ファイル readDataPort_test_cpp.png を追加

- ファイル writeDataPort_test_cpp.png を追加

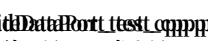
- ファイル writeDataPort_test_cpp.txt を追加

- ファイル writeDataPort_test_cpp.txt を追加

- ファイル readDataPort_test_cpp.txt を追加

- 進捗率 を 50 から 60 に変更

- C++のRTCと接続しての実験
  条件は上の実験と同じ。


突然読み込み、書き込みの時間が激増する現象は発生しませんでした。
以前測ったデータではこれより時間が掛かっていたのですが、OutPort.hでポートプロファイルにデータを書き込む機能を消したら改善しました。

- メモリ使用量

メモリ使用量は以下の通り、rtm.pyについては3時間経過ぐらいまでは増加するものの、その後は7～9MB程度になっているので問題ないと思います。

経過時間	rtm.py	testIn.exe	testOut.exe
0	8.0MB	0.9MB	0.9MB
1時間	18.6MB	0.9MB	0.9MB
3時間	43.9MB	0.9MB	0.9MB
4時間30分	8.2MB	0.9MB	0.9MB
5時間30分	9.2MB	0.9MB	0.9MB
7時間	7.4MB	0.9MB	0.9MB

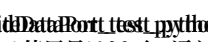
#3 - 2017/06/21 19:13 - n-miyamoto

- ファイル writeDataPort_test_python.png を追加

- ファイル readDataPort_test_python.png を追加

- 進捗率 を 60 から 70 に変更

PythonのRTCを9時間30分程動作させてデータ読み込み、書き込みにかかった時間を計測しました。


メモリ使用量は以下の通りです。

経過時間	rtm.py	testOut	testIn
0	10.8MB	11.8MB	11.8MB
1時間	7.1MB	11.5MB	11.4MB
2時間	9.3MB	9.0MB	8.9MB
3時間	43.9MB	62.1MB	62.1MB
4時間	12.7MB	13.9MB	14.1MB
5時間30分	11.4MB	12.4MB	12.7MB
8時間	14.7MB	122.0MB	13.6MB
9時間30分	12.8MB	12.8MB	12.4MB

3時間後ぐらいにrtm.pyのメモリ使用量が大きくなり、その後起動時のメモリ使用量と同程度に戻るのはC++の実験と同じ傾向です。
ただ、8時間後ぐらいにOutPortを持つRTCのメモリ使用量が激増しています。ただこれも少し時間が経過すれば元に戻っています。

#4 - 2017/06/28 17:30 - n-miyamoto

- ファイル readDataPort_test_python_memory.png を追加

- ファイル writeDataPort_test_python_memory.png を追加

- ファイル memory_py2.txt を追加

- ファイル readDataPort_test_py2.txt を追加

- ファイル writeDataPort_test_py2.txt を追加

- 進捗率 を 70 から 80 に変更

PythonのRTCで再度実験。
今度はメモリ使用量も逐一計測してみました。

readDataPort_test_python_memory.png
writeDataPort_test_python_memory.png
データ読み込み、書き込みの時間が増大したタイミングで、メモリ使用量も突然増えているという結果になりました。

#5 - 2017/07/06 14:46 - n-miyamoto

- ファイル writeDataPort_test_disconnect2.png を追加

- ステータス を 担当 から 解決 に変更

- 進捗率 を 80 から 100 に変更

rtm.pyの変更をプルリクエストしたので、この作業は終了にします。

#6 - 2017/08/30 14:20 - n-ando

- ステータス を 解決 から 終了 に変更

ファイル

data.xlsx	8.64 MB	2017/06/15	n-miyamoto
readDataPort_test.txt	4.5 MB	2017/06/15	n-miyamoto
writeDataPort_test.txt	3.81 MB	2017/06/15	n-miyamoto
readDataPort_test.png	9.21 KB	2017/06/15	n-miyamoto
writeDataPort_test.png	9.79 KB	2017/06/15	n-miyamoto
writeDataPort_test_disconnect.png	13.1 KB	2017/06/15	n-miyamoto
readDataPort_test_disconnect.png	13.1 KB	2017/06/15	n-miyamoto
readDataPort_test_cpp.png	11.3 KB	2017/06/20	n-miyamoto
writeDataPort_test_cpp.png	11 KB	2017/06/20	n-miyamoto
writeDataPort_test_cpp.txt	3.05 MB	2017/06/20	n-miyamoto
writeDataPort_test_cpp.txt	3.05 MB	2017/06/20	n-miyamoto
readDataPort_test_cpp.txt	4.54 MB	2017/06/20	n-miyamoto
writeDataPort_test_python.png	9.67 KB	2017/06/21	n-miyamoto
readDataPort_test_python.png	11.6 KB	2017/06/21	n-miyamoto
readDataPort_test_python_memory.png	21.8 KB	2017/06/28	n-miyamoto

writeDataPort_test_python_memory.png	21.6 KB	2017/06/28	n-miyamoto
memory_py2.txt	1.38 MB	2017/06/28	n-miyamoto
readDataPort_test_py2.txt	1.18 MB	2017/06/28	n-miyamoto
writeDataPort_test_py2.txt	1.14 MB	2017/06/28	n-miyamoto
writeDataPort_test_disconnect2.png	11.1 KB	2017/07/06	n-miyamoto