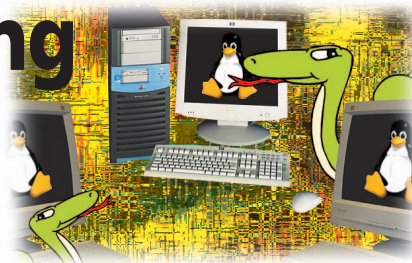


Distributed Programming dengan Python



Untuk menyelesaikan masalah yang rumit, satu komputer saja tidak cukup. Kita dapat meminta komputer-komputer lain untuk membantu kita.

Komputer secara sederhana digunakan untuk membantu memecahkan masalah manusia.

Mulai dari masalah yang sederhana sampai masalah yang kompleks. Pemecahan masalah sederhana tentu mudah saja. Manusia meminta bantuan komputer karena komputer dapat mengerjakannya lebih cepat dan kemungkinan kesalahan lebih kecil. Untuk menyelesaikan masalah sederhana, satu proses atau satu komputer saja sudah dapat diandalkan. Sebagai contoh, menghitung rata-rata penjualan produk di berbagai wilayah di Indonesia. Tanpa komputer pun, masalah tersebut bisa diselesaikan.

Tapi, silakan bayangkan masalah pencarian bilangan prima terbesar. Manusia mungkin dapat memberikan gambaran umum simulasi dengan berbagai tekniknya, atau rumus-rumus yang dibutuhkan, namun, untuk detail dan segala kemungkinannya, sangat susah (dan hampir tidak mungkin) dikerjakan oleh seorang manusia.

Menggunakan satu superkomputer untuk menghitung tentu boleh-boleh saja. Namun, harga satu superkomputer sangatlah mahal. Dan sebenarnya, kita memiliki pilihan lain untuk menyelesaikan masalah tersebut: *distributed programming* atau *distributed application*.

Pengertian distributed berarti suatu sistem di mana tidak semua bagian dikerjakan di dalam *address space* yang sama dan penerapan di dunia nyata umumnya melibatkan lebih dari satu komputer. Setiap komputer, baik yang terhubung sebagai jaringan lokal atau Internet bisa berkontribusi untuk membantu menyelesaikan satu masalah besar.

Namun, sesuatu yang terdistribusi selalu akan membangkitkan masalah lain. Sebut saja konkurensi data, latensi dan berbagai kemungkinan kesalahan lain. Namun,

distributed programming dapat membantu menyelesaikan masalah di dunia nyata dengan *cost* yang sangat masuk akal. Suatu tantangan bagi kita, yang bergerak di dunia TI, untuk menyelesaikan suatu masalah yang solusi yang nyata dan tepat guna.

Dalam suatu *distributed system*, secara sederhana ada sebuah sub-system yang mampu memberikan tugas kepada setiap sistem yang ingin bergabung, dan menerima kembali hasilnya. Yang penting adalah, sebaiknya, kita tidak membatasi sistem operasi atau *platform* setiap sistem yang bergabung. Artinya, kita menggunakan sesuatu yang standar, yang sudah diimplementasikan oleh berbagai platform. Itu kalau kita ingin membuat distributed system yang baik dan dapat digunakan secara meluas.

Yang akan dibahas dalam artikel ini adalah pembuatan sub-system tersebut. Kita telah sepakat bahwa sub-system tersebut mampu melayani sistem dari berbagai platform. Kita akan menggunakan teknologi-teknologi standar. Kita akan mengenal beberapa teknologi tersebut. Pada akhirnya, kita akan memilih satu dan membuat sub-system yang dibicarakan beserta aplikasi *client* yang dijalankan oleh sistem-sistem yang ingin bergabung dengan distributed system kita.

Teknologi-teknologi yang bisa digunakan

Orang selalu mencoba untuk mendapatkan teknologi terbaik untuk membantu menyelesaikan masalah distributed system ini. Sejak awal-awal komputer mulai dikenal sebagai sesuatu yang dapat membantu manusia, berbagai cara dilakukan agar komputer dapat membantu lebih.

Untuk teknologi distributed, kita akan membagi ke dalam beberapa kategori untuk mempermudah. Yang pertama adalah

kategori *socket*. Yang satu ini adalah nenek moyang dari teknologi distributed system. Sebuah server akan membuka socket dan mendefinisikan berbagai aturannya sementara client-client menuruti segala aturannya dan melakukan sesuatu sesuai tugasnya. Dalam konteks distributed, satu atau lebih sub system yang melayani akan membuka socket, dan memungkinkan berbagai sistem lain untuk membantu menyelesaikan masalah.

Menggunakan socket hanya cocok apabila Anda ingin membuat semacam protokol baru. Dan itu sangatlah susah, kompleks, merepotkan dan mungkin *reinventing the wheel*. Apabila problem dapat diselesaikan dengan layer di atas socket, bekerjalah dengan layer tersebut.

Yang kedua adalah *Remote Procedure Call* (RPC). Interaksi dilakukan dengan pemanggilan prosedur. Yang satu ini bukan teknologi baru karena paling tidak, telah dipertimbangkan kurang lebih sekitar tahun 1976. Teknologi ini membutuhkan model pengalamatan server, protokol transport dan *type marshalling*. RPC sendiri memiliki banyak implementasi. Sebut saja Sun RPC, DCE, XML-RPC, dan SOAP.

Yang ketiga adalah *object* terdistribusi namun dalam satu bahasa pemrograman. Contohnya adalah Java Remote Method Invocation (RMI), DOPY, dan Pyro.

Yang keempat adalah object terdistribusi namun tidak membedakan perbedaan bahasa pemrograman. Contohnya adalah DCOM (dari Microsoft) dan CORBA.

Yang kelima adalah teknologi lain seperti *Message-oriented middleware*, *mobile agent*, dan lain sebagainya.

Dari sekian teknologi-teknologi tersebut, yang akan kita pilih dan gunakan dalam artikel ini adalah XML-RPC. Mulai saat ini, setiap pembahasan difokuskan pada XML-RPC, perbandingan dengan teknologi lain, dan implementasi XML-RPC.

```

nopcode@nopcode:~/tutordpyTEST$ python minibank.py
localhost - [23-Feb-2004 02:08:36] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:36] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:36] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:36] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:36] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:37] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:37] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:37] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:37] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:37] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:38] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:38] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:38] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:39] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:39] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:39] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:39] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:39] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:40] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:40] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:40] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:40] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:41] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:41] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:41] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:41] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:42] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:42] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:42] "POST /RPC HTTP/1.0" 200 -
localhost - [23-Feb-2004 02:08:43] "POST /RPC HTTP/1.0" 200 -

```

▲ Server MiniBank sedang melayani client.

Mengapa XML-RPC?

Pertama-tama, kita akan melihat lebih detail tentang XML-RPC, perbandingan dengan teknologi lain dan kebutuhan.

XML-RPC adalah salah satu protokol RPC yang sederhana. Di pembahasan RPC sebelumnya, kita membahas bahwa RPC membutuhkan pengalamatan server, protokol transport, dan type marshalling. Untuk XML-RPC, HTTP digunakan untuk pengalaman server dan protokol transport. Sementara, XML digunakan digunakan untuk type marshalling. Jadi, XML-RPC menggunakan berbagai teknologi yang telah standar.

Tidak heran apabila XML-RPC adalah teknologi yang telah sangat matang. Berbagai bahasa pemrograman juga telah mengimplementasikan teknologi ini.

Berikut ini, kita akan mencoba untuk membandingkan antara XML-RPC dengan beberapa teknologi populer lain.

Pertama, XML-RPC dan CORBA. XML-RPC termasuk prosedural, sementara CORBA object oriented. Untuk transmisi data, XML-RPC menggunakan XML, sementara CORBA menggunakan format binary. Secara umum, CORBA lebih pantas digunakan pada aplikasi *enterprise*.

Yang kedua adalah XML-RPC dan SOAP. SOAP sendiri sangatlah mirip dengan XML-RPC. SOAP juga menggunakan HTTP dan XML. SOAP lebih kompleks dibandingkan XML-RPC.

Sayangnya, SOAP masih jauh dari stabil.

Baik XML-RPC, SOAP ataupun CORBA memiliki implementasi yang matang untuk

Python. Terdapat banyak implementasi XML-RPC, SOAP ataupun CORBA untuk bahasa Python. Beberapa lebih stabil, banyak fitur dan lebih di-maintain. Khusus untuk XML-RPC, mulai Python 2.2, implementasi oleh Pythonware (<http://www.pythonware.com/products/xmlrpc/>), yaitu xmlrpclib telah dimasukkan ke dalam distribusi resmi Python. Dengan distribusi resmi Python saja, kita dapat menulis XML-RPC server dan client dengan sangat mudah. Implementasi SOAP yang terkenal antara lain SOAP.py (pywebsvcs.sf.net), mirip dengan xmlrpclib, ZSI (Zolera SOAP Infrastructure, pywebsvcs.sf.net), SOAPy (soapy.sf.net) dan 4Suite SOAP (bagian dari 4Suite server, 4suite.org).

Alasan mengapa kita menggunakan XML-RPC adalah pertama, sesuai dengan kebutuhan. Dalam berbagai kasus, XML-RPC dapat diandalkan untuk distributed system. Penulis memanfaatkan XML-RPC untuk proyek-proyek software (cukup besar) yang penulis kerjakan dan XML-RPC dapat diandalkan. Yang kedua adalah stabil. XML-RPC bahkan dinyatakan mungkin terlalu stabil. Yang ketiga adalah mudah digunakan. Dibandingkan dengan CORBA misalnya, XML-RPC jelas jauh lebih sederhana dan mudah digunakan. Yang keempat adalah, segala kebutuhan kita telah termasuk dalam distribusi Python sejak versi 2.2.

Bagi Anda yang bekerja untuk sistem enterprise dan sangat sangat kompleks, pertimbangkan CORBA. XML-RPC memiliki sejumlah kekurangan apabila dibandingkan

dengan CORBA. CORBA secara umum adalah teknologi yang sangat baik.

Beberapa contoh sukses pemanfaatan XML-RPC di dunia free software adalah KDE dan Zope. KDE datang dengan kxmlrpcd, yang memungkinkan kita melakukan mengatur aplikasi KDE dengan XML-RPC. Dengan menggunakan XML-RPC, kita dapat memanipulasi berbagai hal seperti Address Book KDE dan KDE Trader. Sementara, Zope sendiri memiliki implementasi XML-RPC yang telah sangat matang.

Bagi Anda yang ingin mengetahui XML-RPC lebih lanjut, Anda dapat membacanya di www.xmlrpc.com. Selanjutnya, kita hanya akan memfokuskan diri untuk menulis server dan client, kemudian membahas sedikit bagaimana XML-RPC dapat digunakan di dunia nyata, baik untuk aplikasi besar ataupun menyelesaikan masalah besar tertentu dengan distributed system.

Client XML-RPC untuk MiniBank

Segala yang Anda butuhkan untuk membuat client XML-RPC dengan Python adalah modul xmlrpclib yang dibuat oleh Pythonware. Itu saja. Dan pembuatan client dapat dilakukan dengan sangat mudah.

Untuk mencoba client, kita membutuhkan sebuah server. Dalam artikel ini, kita akan menggunakan server <http://nop.keant.org:2700/>. Source code server dibahas di bagian pembuatan server. Apabila Anda tidak memiliki koneksi internet, Anda dapat membuat terlebih dahulu server-nya dan menjalankannya. Di

```

>>> bank = xmlrpclib.Server('http://localhost:2700/')
>>> bank.register('Nop!', '12345', 'Jakarta', 'nopcode', 'Superman')
1077476997
>>> bank.getInfo(1077476997)
{'1077476997', 'Nop!', '12345', 'Jakarta', 'nopcode', 'Superman'}
>>> bank.getmount(1077476997)
0
>>> bank.credit(1077476997, 100000000)
0
>>> bank.credit(1077476997, 100000000)
0
>>> bank.credit(1077476997, 100000000)
0
>>> bank.credit(1077476997, 100000000)
0
>>> bank.getmount(1077476997)
40000000
>>> bank.debit(1077476997, 100000000)
0
>>> bank.getmount(1077476997)
30000000
>>> bank.debit(1077476997, 100000000)
0
>>> bank.debit(1077476997, 100000000)
0
>>> bank.debit(1077476997, 100000000)
0
>>> bank.getmount(1077476997)
10000000
>>>

```

▲ Client MiniBank mendapatkan informasi dari Server MiniBank.

```
nop@nop:~$ python2.3 xmlrpccli.py
>>> bank = xmlrpccli.Server('http://localhost:2700')
>>> bank.Register('Nopri', '12345', 'Jakarta', 'nop@nop', 'Superman')
Traceback (most recent call last):
  File "xmlrpccli.py", line 1029, in __call__
    return self._send(self._name, args)
  File "xmlrpccli.py", line 1316, in _request
    verbose=self._verbose
  File "xmlrpccli.py", line 1064, in request
    errcode, errmsg, headers = b.getreply()
  File "xmlrpccli.py", line 1025, in getreply
    response = self._conn.getresponse()
  File "xmlrpccli.py", line 779, in getresponse
    response.begin()
  File "xmlrpccli.py", line 273, in begin
    version, status, reason = self._read_status()
  File "xmlrpccli.py", line 231, in _read_status
    line = self._fp.readline()
  File "xmlrpccli.py", line 323, in readline
    data = recu()
socket.error: (104, "Connection reset by peer")
>>>
```

▲ MiniBank2 menolak client dari alamat IP tertentu.

dalam server `http://nop.keant.org:2700` tersebut, penulis meletakkan XML-RPC server untuk mencoba XML-RPC client yang akan kita buat. Server yang dimaksud adalah simulasi sebuah Bank sederhana MiniBank yang mampu melakukan pendaftaran nasabah baru, nasabah mengambil dan menyetorkan tabungan, pencetakan informasi nasabah dan informasi saldo nasabah. Tentunya, semua ini hanyalah simulasi dan sangat tidak sesuai dengan keadaan perbankan sebenarnya. Informasi disimpan di dalam memory, sehingga apabila server dihentikan atau di-restart, semua informasi akan hilang.

Berikut ini adalah langkah-langkah untuk membuat client:

- Import module `xmlrpccli`

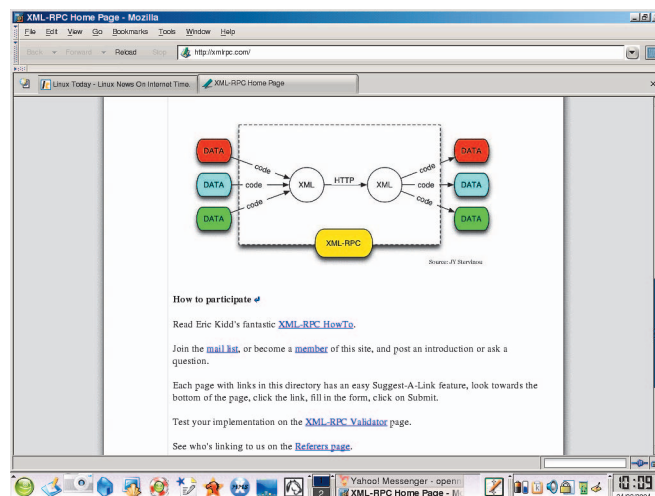
```
import xmlrpccli
```

- Lakukan koneksi ke server

```
bank = xmlrpccli.Server('http://
nop.keant.org:2700')
```

Untuk selanjutnya, Anda dapat langsung memanggil method-method class yang diekspor seperti contoh berikut:

```
>>> bank.Register('Nopri', '12345',
'Jakarta', 'nop@nop', 'Superman')
1077470785
>>> bank.Register('Anto', '23456',
'Jakarta', 'nop2@nop', 'Superman2')
1077470800
>>> bank.GetInfo(1077470785)
[1077470785, 'Nopri', '12345', 'Jakarta',
'nop@nop', 'Superman']
```



▲ Bagan cara kerja XMLRPC.

- CUSTOMERS, berguna untuk menampung data para nasabah.
- AMOUNT, berguna untuk menampung informasi saldo nasabah.

Sementara, berikut ini adalah method yang dimiliki oleh MiniBank:

- Register(), berguna untuk mendaftarkan nasabah baru.
- Credit(), berguna untuk menyetorkan uang tabungan nasabah.
- Debit(), berguna untuk mengambil tabungan.
- GetInfo(), berguna untuk menampilkan informasi nasabah.
- GetAmount(), berguna untuk mendapatkan saldo nasabah.
- Help(), petunjuk bagi operator bank.

Class ini memiliki constructor, yang berguna untuk mengatur nilai properti-properti MiniBank.

Berikut ini adalah langkah-langkah untuk membuat server:

- Siapkan class MiniBank.
- Buat server baru.
- Registerkan class MiniBank.
- Jalankan server.

Begitu mudah. Berikut ini, kita akan melihat source code sepenuhnya dari server MiniBank, `minibank.py`:

```
#!/c) Noprianto, 22 Feb 2004, for
educational purpose only
```

```
import SimpleXMLRPCServer
import time
```

```
>>> bank.GetAmount(1077470785)
0
>>> bank.Credit(1077470785,1000000)
0
>>> bank.GetAmount(1077470785)
1000000
>>> bank.Debit(1077470785,5000000)
-2
>>> bank.GetAmount(1077470785)
1000000
>>> bank.Debit(1077470785,50000)
0
>>> bank.GetAmount(1077470785)
950000
```

Dengan hanya beberapa baris code, Anda telah membuat sebuah XML-RPC client. XML-RPC server yang baik tentunya hanya memberikan nilai, sehingga Anda bisa memformat kembali nilai yang dikembalikan dalam tampilan yang lebih indah.

MiniBank: Server XML-RPC

Apa yang Anda butuhkan dalam membuat sebuah XML-RPC server adalah modul SimpleXMLRPCServer. Pembuatan server dapat dilakukan dengan sangat mudah.

Dalam kasus MiniBank, kita memiliki sebuah class dengan nama MiniBank. Class tersebut memiliki properti sebagai berikut:

```

class MiniBank:
    def __init__(self):
        """
        Constructor for Class MiniBank
        """
        self.CUSTOMERS = []
        self.AMOUNT = []

    def Register(self, name, id, address,
        email, extra_info):
        """
        Register new account. Returns
        account_ID
        """
        unique = 0
        while not unique:
            account_ID = int(time.time())
            unique = 1
            for i in self.CUSTOMERS:
                if account_ID == i[0]:
                    unique = 0
                    break

            customer = list([account_ID,
                name, id, address, email,
                extra_info])
            customer_amount = list
            ([account_ID, 0])
            self.CUSTOMERS.append
            (customer)
            self.AMOUNT.append(customer_
            amount)
            return account_ID;

    def Credit(self, account_ID, amount):
        """
        Credit ...
        return value code:
        0 = success
        -1 = failed, because we cant
        find the customer
        """
        code = -1
        for i in self.AMOUNT:
            if i[0] == account_ID:
                i[1] += amount
                code = 0
                break
        return code

```

```

def Debit(self, account_ID, amount):
    """
    Debit ...
    return value code:
    0 = success
    -1 = failed, because we cant
    find the customer
    -2 = failed, because the
    customer has no enough money
    """
    code = -1
    for i in self.AMOUNT:
        if i[0] == account_ID:
            if amount >= i[1]:
                code = -2
                break
            else:
                i[1] -= amount
                code = 0
                break
    return code

def GetAmount(self, account_ID):
    """
    Get Amount ...
    return value code:
    amount of money = success
    -1 = failed, because we cant
    find the customer
    """
    code = -1
    for i in self.AMOUNT:
        if i[0] == account_ID:
            code = i[1]
            break
    return code

def GetInfo(self, account_ID):
    """
    Get Info ...
    return value code:
    info = success
    -1 = failed, because we cant
    find the customer
    """
    code = -1
    for i in self.CUSTOMERS:
        if i[0] == account_ID:
            code = i
            break
    return code

def Help(self):
    """

```

```

Help for internals
"""
return """
Steps to have a bank account at
MiniBank:
0. We need name, id, address,
email...
1. Ask information from new
customer, and we fill the
informations into database.
2. Customer prepares some
money, at least Rp. 100.000.
Save it.
3. Give the MiniBank Card
(which is automatically printed)
to customer.
4. Save the money into
appropriate place.

Steps to do Credit:
0. We need account_ID
1. Call Credit(account_ID, amount)

Steps to do Debit:
0. We need account_ID
1. Call Debit(account_ID, amount)

Steps to get amount:
0. We need account_ID
1. Call GetAmount(account_ID)

Steps to get info:
0. We need account_ID
1. Call GetInfo(account_ID)

"""

server = SimpleXMLRPCServer.
SimpleXMLRPCServer(('localhost',2700))
server.register_instance(MiniBank())
server.serve_forever()

```

Selesai. Apabila diperhatikan, source code sepanjang kurang lebih 2.8 K didominasi oleh code-code untuk class MiniBank. Untuk pembuatan server-nya, kita hanya membutuhkan 3 baris code.

Catatan: Penjelasan untuk method Register(). Nomor rekening nasabah (account_ID) didapat dari waktu sistem. Tentunya, mungkin sekali terdapat banyak pendaftar dalam satu detik yang sama. Dan sangat berisiko untuk begitu saja mempercayakan

nomor rekening pada waktu sistem. Untuk itu, kita melakukan pemeriksaan sederhana. Pertama-tama, asumsikan kita tidak mendapatkan nomor rekening yang unik. Kita akan terus mengulang sampai mendapatkan nomor rekening yang unik. Setiap kali perulangan, kita mencoba mendapatkan waktu sistem dan sekaligus memeriksa apakah nomor rekening tersebut sudah dipakai. Apabila ternyata telah terpakai, maka kita mendapatkan kembali waktu sistem. Demikian seterusnya sampai nomor rekening unik didapatkan.

MiniBank2: MiniBank yang lebih canggih

MiniBank kita sebelumnya memiliki sedikit kekurangan. Cobalah menjalankan minibank.py dan tekanlah kombinasi tombol CTRL-C. Langsung jalankan kembali minibank.py. Sebuah pesan kesalahan socket yang mengatakan bahwa alamat sedang terpakai akan muncul.

Kemudian, semua pihak bisa mengakses minibank.py. Selain dari sisi sistem operasi, sebenarnya kita dapat membuat sebuah access list sendiri yang sederhana, yang berisikan alamat IP mana saja yang dapat mengakses server kita.

Untuk itu, beberapa perbaikan kita lakukan. Simpanlah dengan nama baru, minibank2.py. Pertama-tama, import-lah sebuah modul tambahan, yaitu modul socket. Setelah itu, tambahkan deklarasi class MiniBankXMLRPCServer di atas deklarasi class MiniBank.

```
class MiniBankXMLRPCServer(SimpleXMLRPCServer.SimpleXMLRPCServer):
    def __init__(self, *args):
        """
        Constructor for Class
        MiniBankXMLRPCServer
        """
    SimpleXMLRPCServer.SimpleXMLRPC
    Server.__init__(self, (args[0], args[1]))
    self.ACCESSLIST = (
        '127.0.0.1',
    )
    def server_bind(self):
        """
        Server bind...
        """
    self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
SOL_SOCKET, socket.SO_
REUSEADDR, 1)
SimpleXMLRPCServer.SimpleXMLRPC
Server.server_bind(self)
def verify_request(self, request,
client_address):
    """
    Verify request, only access IP in
    ACCESSLIST
    """
    if client_address[0] in self.
ACCESSLIST:
        return 1
    else:
        return 0
```

Yang terakhir, gantilah baris:

```
server = SimpleXMLRPCServer.Simple
XMLRPCServer(('localhost',2700))
```

Dengan baris berikut ini:

```
server = MiniBankXMLRPCServer
('localhost',2800)
```

Server minibank2.py yang dijalankan pada port 2800 hanya menerima koneksi dari lokal. Untuk mengubahnya, gantilah properti ACCESSLIST dari class MiniBankXMLRPCServer.

Sebenarnya, apakah yang kita lakukan sampai Minibank2 kita mampu menggunakan kembali port yang telah digunakan dan mampu membatasi koneksi dari IP tertentu? Apa yang kita lakukan adalah membuat class MiniBankXMLRPCServer yang merupakan turunan dari class SimpleXMLRPCServer. Setelah itu, kita mengubah beberapa method-nya.

Di bagian bawah program, ketika membuat server, kita tidak lagi menggunakan SimpleXMLRPCServer.SimpleXMLRPCServer, melainkan MiniBankXMLRPCServer.

Berbagai modifikasi dapat dilakukan agar kita mendapatkan XML-RPC server yang lebih baik. Anda yang mengerti perbankan tentunya dapat mengembangkan perbankan yang lebih serius. Siapa tahu terdapat banyak bank yang ingin membeli sistem yang kita kembangkan.

Penggunaan di dunia nyata

XML-RPC digunakan secara meluas.


Termasuk oleh Red Hat baik dalam beberapa

layanan mereka ataupun secara internal. Contoh lain, penulis juga terlibat aktif dalam pengembangan aplikasi bisnis dengan pemanfaatan XML-RPC secara intensif.

Sekarang, kita akan melihat bagaimana XML-RPC dapat membantu kita, dalam kasus sederhana MiniBank. Kita dapat membuat layanan perbankan MiniBank dengan cara biasa, atau dengan pemrograman web biasa. Namun, yang dapat mengaksesnya hanyalah script atau HTML dalam aplikasi kita. Dengan penggunaan XML-RPC, MiniBank dapat diakses dari web, dari desktop seperti yang telah kita lakukan sebelumnya ataupun dari berbagai client yang mendukung XML-RPC.

Kasus lain. Seandainya saja Anda ingin melakukan komputasi besar yang tidak dapat dikerjakan sendiri. Anda dapat saja membuat sebuah XML-RPC server yang dapat memberikan tugas tertentu dan menerima kembali hasilnya. Tentunya, alangkah baiknya kalau Anda juga membuat clientnya. Hasil pengolahan dari client-client di seluruh dunia dapat Anda kumpulkan. Hanya, seperti halnya distributed system, Anda akan menghadapi masalah latensi, konkurensi data dan kemungkinan kesalahan.

XML-RPC adalah teknologi yang cukup bagus. Dan sekaligus tepat digunakan dalam membuat distributed system. Sekali lagi, ide distributed system adalah ide yang menarik. Sebuah superkomputer tidak mungkin terjangkau oleh pengguna komputer rumahan yang memiliki ide luar biasa. Karena di dunia ini terdapat lebih dari satu komputer dan kita memiliki jaringan internet yang dapat diakses oleh penduduk dunia, maka daripada membeli sebuah super komputer, Anda dapat meminta bantuan para pengguna komputer di seluruh dunia.

Tentunya, pengguna komputer lain yang membantu Anda harus tetap dapat bekerja seperti biasanya. Komputernya akan membantu Anda ketika sedang idle, atau Anda dapat membuat client yang berwujud game. Sambil dimainkan, pekerjaan lain untuk Anda dikerjakan. Apabila terhubung ke internet, client dapat mengirimkan hasil pekerjaannya kepada Anda. Sama-sama senang. Selamat mencoba! 

Noprianto (noprianto@infolinux.co.id)