CHAPTER 1

Basys Network

This chapter presents the Basys package. Basys implements an abstract layer for networks which require bidirectional communication between clients and servers. Basys manages connections in such a way that sending and receiving data can be performed independently and simultaneously.

1.1 Basics

Each Basys connection runs its own incoming data loop which processes received data asynchronously. Every received data is processed in a separate thread. Communication between client and server are equivalent in both directions. There is only one difference: a server should accept connection from a client and it cannot establish new one by itself.

Basys models a network as connected peers (as shown by Figure 1.1). Each peer (network node) has its own `BasysNetwork` instance. `BasysNetwork` is

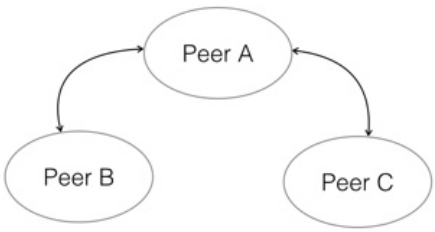

Figure 1.1 A network is a graph of peers.

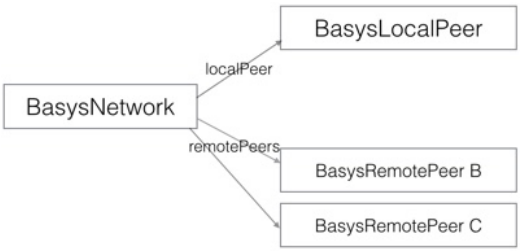

Figure 1.2 A peer has a local and remote peers.

main class of Basys and should be subclassed by concrete network implementations.

Example

In Basys, to represent a network with multiple hosts, we create one instance of `BasysNetwork` (subclasses since `BasysNetwork` is an abstract class specifying the hooks that should be specialized by the application using Basys) for each node in the network.

For example imagine that we want to run a server on port 40433, On a server side, we will start a basys server on a network instance as follows:

```
network := BasysNetwork new.
network startServerOn: 40433
```

The server will start to listen for incoming client connections.

Then on a client we define a new BasysNetwork and connect it to the running server using the message `remotePeerAt:` as follows:

```
network := BasysNetwork new.
network remotePeerAt: (TCPAddress localAt: 40433)
```

1.2 Network peers

A `BasysNetwork` instance has two main instance variables:

- `localPeer`: One local peer (instance of `BasysLocalPeer`) which represents current local peer. Its purpose is to provide some id to be identified between other network peers.
- `remotePeers`: Network remote peers is a collection of `BasysRemotePeer` instances which represent the currently connected remote peers (as shown in Figure 1.3).

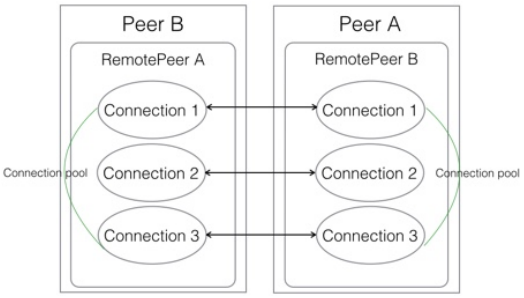

Figure 1.3 On location B we have a remote peer instance representing location A and vice versa.

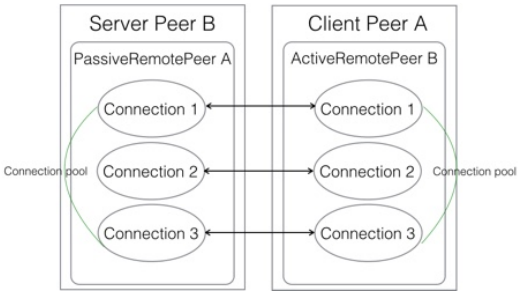

Figure 1.4

On location B we have a remote peer instance representing location A and vice versa (as shown in Figure 1.3). Each remote peer maintains a pool of connections with the remote side. For each communication between different computers, we have a separate remote peer with its set of connections: each connection in a remote peer will point to the same remote peer.

Any established connection on client is added to peer which represents server. And any established connection to server is added to peer which represents client. Peers interact with each others using connection pools.

To interact with remote side, a peer retrieves a free connection from pool or establish new if it can. Then the retrieved connection is passed to a given action block using the message `execute:`

```
peer execute: [ :connection | connection sendDataPacket: dataObject
    ].
    "or short version"
peer sendDataPacket: dataObject
```

There are two kinds of remote peers: active (`BasysActiveRemotePeer`) and passive (`BasysPassiveRemotePeer`).

Active peers. Active peers are created by client for a given address using the message `remotePeerAt:` as follows:

```
peer := network remotePeerAt: (TCPAddress localAt: 40433)
```

They establish new connections on demand when no one exists in pool or all are busy.

Passive peers. Passive peers are created by a `BasysServer` for incoming connections using the message `startServerOn:`. Servers should be started on network as follows:

```
network startServerOn: 40433
```

To perform requests, passive peers are waiting for free connections. They cannot create new connection but have to wait when all connections are busy or none exists.

1.3 Peers identification

Basys performs identification procedure for all established connections. It should ensure that all client connections belong to same peer instance on server. This is what is done in the method `establishNewConnection`:

```
BasysActivePeer >> establishNewConnection
   connection := network establishNewConnectionTo: address.
   connection acceptIncomingData.
   [ receivedRemotePeerId := network identifyLocalPeerOn: connection.
   self assignId: receivedRemotePeerId receivedFrom: connection ]
    on: Error
  do: [:err | connection close.
      err pass ].
   ^ connection
```

The protocol for peer identification (message `identifyLocalPeerOn:`) is the responsibility of concrete network implementations.

On server side

For every new connection, a `BasysServer` asks its unique network to create a new passive peer and sends incoming connection to it. Then a client initiates identification procedure for this connection. It should take care of three cases:

- A server should detect that new connection belongs to existing peer. In that case connection should be migrated to the existing peer and anonymous peer should be removed.
- A server should detect that new peer is connected. Then it should assign to it a correct id which was received from connected client peer.

- At the end, the server should send to the client the server local id.

For the first case, passive peer defines a convenient method named `beIdentifiedAs:`

```
BasysPassivePeer >> beIdentifiedAs: peerId
  | registeredPeer |
  registeredPeer := network
            remotePeerWithId: peerId
            ifAbsent: [ id := peerId. ^self ].
  self == registeredPeer ifTrue: [ self error: 'Should not happen' ].
  registeredPeer importConnectionsFrom: connectionPool.
  network removeRemotePeer: self.
  ^ registeredPeer
```

On client side

A client establishes a new connection to a server. Then it initiates the identification procedure. At the end, it receives the server id. There are two cases here:

- the received peer id belongs to an existing passive peer (if client is server itself and it already accepted connections from server),
- there is no existing peers with received id.

In first case established connection will be migrated to the existing peer. The existing peer will become active and the original peer will be removed.

In last case, the received id will be assigned to the original peer as shown by method `assignId:receivedFrom:`:

```
BasysActivePeer >> assignId: peerId receivedFrom: aBasysConnection
  | registeredPeer |
  id ifNotNil: [
    id = peerId ifFalse: [ self error: 'Peer should not change id
    when it was already identified' ].
    ^ self ].
  registeredPeer := network remotePeerWithId: peerId ifAbsent: [ ^id
    := peerId ].
  registeredPeer addNewConnection: aBasysConnection.
  registeredPeer becomeActiveToReplaceSamePeer: self
```

1.4 Basys concretisation

Each application that wants to take advantage of Basys should specialize five hooks. Basys implementors should subclass the class `BasysNetwork` and answer the following five questions by defining the corresponding methods:

- What is network data?

- What to do with data?

```
process: dataObject receivedFrom: senderPeer
```

- How to send data?

```
sendDataPacket: dataObject by: aBasysConnection
```

- How to receive data?

```
receiveIncomingDataPacketFrom: senderPeer by: aBasysConnection
```

- How to identify peers?

```
identifyLocalPeerOn: aBasysConnection
```

Seamless implements these operations to create transparent network of distributed objects.