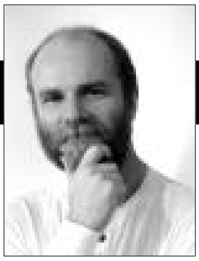
Managing Objects


Jan Steinman

Barbara Yates

Smalltalk as an Internet server

IN THE JUNE ISSUE, we demonstrated how to turn arbitrary *Text* objects into HTML, and in September 1995, we demonstrated how to modify VisualWorks under ENVY so you could store all your commentary in styled text. The only thing missing to have live web access to your Smalltalk project documentation is a server!

We've never been fond of specific solutions to general problems. It would be easy to hard-code a server that is dedicated to serving HTML versions of Smalltalk documentation, but there is so much code that is common to *any* server that we couldn't ignore the potential reuse.

For example, a server of any kind has these needs:

- Manage a socket name space — you can't simply pick any number for your socket.
- Initialize a socket and prepare it for use.
- Loop forever, waiting for connection requests.
- Record service requests in a log.
- Screen service requests for security reasons.
- Fork off individual service requests, so the main loop isn't unduly delayed from its primary task of waiting for connections.
- Manage unexpected problems that might occur in a service request.
- Perform the requested service, and return a result.

The following TCP server framework for VisualWorks hides all but the last step, allowing the server author to concentrate on the actual service being provided, without being distracted by the mechanics of managing sockets, processes, logs, and exceptions.

SETTING UP A SERVER INSTANCE

Our server is defined as:

```
Object subclass: #TcpServer
    instanceVariableNames: 'port socket server service
      handler requests logger logProtect'
    classVariableNames: 'CanTalkToBlock
      DefaultHandlers DefaultServices Portmap
      PortmapControl '
    poolDictionaries: ''
```

Instances of *TcpServer* provide stateless services on Transmission Control Protocol (TCP) Internet domain sockets. Each instance is uniquely associated with a port number on a given machine, which must be supplied when creating an instance. Because port must be unique, we use a class Portmap registry to maintain this uniqueness. Because this registry will be accessed from multiple threads of control, it must be protected by a mutual exclusion mechanism in PortmapControl. We set all this up when initializing the class, which also sets up the default security and the DefaultHandlers and DefaultServices.

```
TcpServer class:
  initialize
    "Set up long-term state that is used for instance
     management."

    self beSecure.
    "If this is a re-initialize, be thread-safe."
    PortmapControl == nil ifFalse: [self shutDown].
    PortmapControl := Semaphore forMutualExclusion.
    Portmap size > 0 ifTrue:
        [Portmap copy do: [:server | server terminate]].
    Portmap := IdentityDictionary new.
    (DefaultHandlers := IdentityDictionary new) at: 0
     put: self nullHandler.
    (DefaultServices := IdentityDictionary new) at: 0
     put: self discardService

  onPort: portNumber
    "Answer an active instance that provides default
     services for port <portNumber>."

    ^Portmap
        at: portNumber asInteger
        ifAbsent: [(self new port: portNumber asInteger)
                    resume]
```

The connection security mechanism employs a block that answers a Boolean when passed an incoming sock-

Jan Steinman and Barbara Yates are co-founders of Bytesmiths, a technical services company that has been helping companies adopt Smalltalk since 1987. Between them, they have more than 22 years of Smalltalk experience. They can be reached at Barbara@Bytesmiths.com or Jan@Bytesmiths.com, or via http://www.bytesmiths.com.

et. If the block answers false, the connection is dropped immediately. This class method sets the default connection security, but once the connection security is passed, an individual server can take extra precautions or implement finer graduations of security. We also have utility methods **beFriendly**, which allows all connections, and **beLonely**, which only allows connections from the same machine, which can be useful for testing.

```
TcpServer class:
    beSecure
        "Set the default security to only accept connections
         from the same network."

        CanTalkToBlock := [:socket |
            socket getPeer networkAddress = socket
            getName networkAddress]
```

The final part of class initialization declares what to do when an instance is created for a port number that does not have a default service block or exception handler. Normally, an instance has its own service and handler. If not, a default service and/or handler is fetched from the class for a given port number. If even that fails, then the "default default" service and/or handler is used. Because zero is an illegal port number, we use it to hold the "default default" service and handler.

```
TcpServer class:
    nullHandler
        "Answer a handler for when nothing is to be done
         with errors. (This is generally not a wise choice!!)"

        ^[:exception :stream | ]

    discardService
        "Answer a 'discard' service, which is to be used
         when no service can be found for a given port."

        ^[:stream | ]
```

Finally, accessing methods for the unique identifying information for an instance must take some special actions.

```
TcpServer:
    port
        "Answer the port that is listened to by this server for
         requests. If none is given, answer 7, for an echo
         server."

        ^port ? [7]

    port: portNumber
        "Initialize me with state needed for default
         communication on the given <portNumber>.
         Answer myself."

        self port: portNumber service: nil handler: nil
                logger: System errorLog
```

The definition of the ? method was published in our January 1996 column. It simply answers the receiver, unless it is nil, in which case the argument is evaluated and answered. Note that we also use a few ENVY utility methods in this code, which you will need to change if you are not going to use this as an ENVY documentation server.

With one more method, we will have all the essential base state needed to implement our server. This is the primary instance initialization method.

```
TcpServer:
    port: portNumber service: serviceBlock
            handler: exceptionBlock logger: logStream
        "Initialize me with state needed for a particular
         task. Any argument can be nil, and will be
         defaulted suitable for an 'echo' server that logs to
         the Transcript.
        <portNumber> is an Integer port number to listen
         to.
        <serviceBlock> is a one-argument block that is
          passed a stream on a transient socket on
          <portNumber> when a connection arrives.
        <exceptionBlock> is a two-argument block that is
          passed the exception and the socket stream when
          <serviceBlock> has an unhandled exception.
        <logStream> is place to write log messages."

        requests := WeakArray with: 0.
        service := serviceBlock.
        handler := exceptionBlock.
        port := portNumber asInteger.
        logger := logStream ? [Transcript].
        (self class register: self) ifFalse:
            [self error: 'You already have a service on this
                        port! You can only have one service
                        per port per machine.']
```

Remember the need to keep track of port numbers? This is handled by the class, which also needs a way to "forget" about port numbers as their server instances are released. The class also manages associations between port numbers and the services (and their exception handlers) that each port provides.

```
TcpServer class:
    register: instance
        "Register the given <instance> of myself, unless one
         is already registered at that port.
        Answer success or failure."

        ^(Portmap at: instance port ifAbsentPut:
           [PortmapControl critical: [instance]]) == instance

    unregister: instance

        "Unregister the given <instance> of myself. Don't
         complain if I can't find it."

        PortmapControl critical:
            [Portmap removeKey: (Portmap keyAtValue:
             instance ifAbsent: []) ifAbsent: []]
    defaultHandlerFor: portNumber
        "Answer an appropriate handler for <portNumber>,
         or a default default if none."

        ^DefaultHandlers at: portNumber ifAbsent:
          [DefaultHandlers at: 0]

    defaultServiceFor: portNumber
        "Answer an appropriate service for <portNumber>,
         or a default default if none."

        ^DefaultServices at: portNumber ifAbsent:
          [DefaultServices at: 0]
```

Recall that class initialization set up a **nullHandler** and a **discardService** to be used when nothing else was specified for a given instance on a given port number. That means we need a way to associate other handlers and services with ports, so that instances can be tightly cohesive with a port number, but loosely coupled with a service and handler.

The "default default" of a **discardService** with a **nullHandler** doesn't make for a very useful server!

```
  TcpServer class:
    defaultHandlerFor: portNumber is: twoArgBlock
        "Set the exception handler for <portNumber> to
         <twoArgBlock>. When evaluated, the two
         arguments will be:
        the <exception> that was the argument to the
        handle: block,
        a read-append <stream> on the transient socket
        that is being serviced.
        This is not thread safe, and should not be changed
        by some server action."

        2 = twoArgBlock numArgs
            ifFalse: [self error: 'Sorry, I need a two-argument
                        clean block here!!']
            ifTrue: [DefaultHandlers at: portNumber put:
                        twoArgBlock]

    defaultServiceFor: portNumber is: oneArgBlock
        "Set the service for <portNumber> to a clean
         <oneArgBlock>. When evaluated, the argument will
         be a read-append <stream> on the transient socket
         that is being serviced."

        1 = oneArgBlock numArgs
            ifFalse: [self error: 'Sorry, I need a one-argument
                        clean block here!!']
            ifTrue: [DefaultServices at: portNumber put:
                        oneArgBlock]
```

MAKING A SERVER INSTANCE USEFUL

Although you need more code for a functional server, we now have the base state needed to create and initialize a server instance. Let's put it to work by deriving the instance state needed, such as the socket connection and input process.

The basic service and handler are usually initialized from the class registry of default services and default handlers:

```
  TcpServer:
    handler
        "Answer a two-argument block that is evaluated
         upon service exception. It is passed the exception
         and a stream. Non-local returns should not be
         attempted. The block answer is discarded.
        If no handler exists, get one suitable for my port."

        ^handler ? [handler := self class defaultHandlerFor:
                              self port]

    service
        "Answer a one-argument clean block that is forked
         upon service request. It is passed a stream on the
         transient socket connection. Stream closing will be
         handled by the evaluator. The block answer is
         discarded.
        If no service has been set, initialize it to one
        appropriate for my port."

        ^service ? [service := self class defaultServiceFor:
                              self port]
```

This class–instance collaboration might not seem necessary; in fact it isn't. However, the temporality of server instances is very different from that of port–service associations, so it's useful to bind server instances tightly to a port number, but loosely to a service.

For example, an instance that is providing World Wide Web service using Hyper Text Transfer Protocol (HTTP) is created and discarded more frequently than the binding of this service to port number 80, the default HTTP port number. This reduces coupling in the time domain, which is often overlooked by designers who concentrate on reducing behavioral or implementation coupling.

Now that we have a service and a handler, the important stuff can happen. An independent thread runs the primary server loop that waits on socket connections, checks to see if the connection is legal, then services the connection's request.

```
  TcpServer:
    server
        "Answer an unscheduled Process that listens for and
         dispatches service requests.
        This service loop should spend most of its time
        waiting on a socket connection, and so has a high
        priority. The service it implements is typically time-
        consuming, and so should be forked at a low
        priority, which immediately allows the server to
        resume listening for connections."

        ^server ? [server := [[ | connection |
            (CanTalkToBlock value: (connection := self
                                        socket accept))
            ifTrue: [self serviceRequest: connection
                        readAppendStream]
            ifFalse: [[connection close] fork]] repeat]
                        newProcess.
        server priority: Processor userInterruptPriority - 1.
        server]
```

```
socket
    "Answer an IOSocketAccessor that listens to my port
     for service requests."

    ^socket ?
        [socket := OSErrorHolder existingReferentSignal
            handle: [:ex |
                logger == nil ifFalse:
                    [logger cr; show: 'You appear to have
                     another server running on port ', self
                     port printString, ' on this machine.']].
                ex returnWith: nil]
            do: [IOAccessor defaultForIPC
                    newTCPserverAtPort: self port]]
```

```
serviceRequest: stream
    "A connection has been accepted on a transient copy
     of my socket. <stream> is a read-write stream on that
     socket. Log the activity and provide the requested
     service in a separate thread at low priority."

    self registerRequest: ([self serviceRequestFork:
                                    stream]
        forkAt: Processor userBackgroundPriority + 1)
```

```
serviceRequestFork: stream
    "Provide a requested service, based on the socket
     <stream>, which at this point has not been read at
     all. If there is a problem, invoke an instance-specific
     handler. This method is forked at a low priority."

    Signal noHandlerSignal
        handle: [:ex | self handler value: ex value: stream]
        do: [self log: 'Open connection at ',
                EmTimeStamp now printString from: stream.
    self service value: stream.
    self log: 'Close connection at ' EmTimeStamp now
                printString from: stream].
    OSErrorHolder errorSignal handle: [:ex |] do:
                                        [stream close]
```

Because individual requests are forked off, it is essential to track them down and kill them if needed, so request threads are registered in requests, a WeakArray. As these requests terminate, they are collected as garbage and removed from requests.

```
TcpServer:
    registerRequest: serviceRequestProcess
        "A connection has been accepted and
         <serviceRequestProcess> has been forked to deal
         with it. Hang onto it weakly, so it can be killed when
         I'm killed. When it terminates, the scavenger will
         remove it from the registry."
```

INFO@SIGS

SIGS Publications, Inc., 71 West 23rd Street, 3rd Floor, New York, NY 10010; 212.242.7447; Fax: 212.242.7574

ARTICLE SUBMISSION

To submit articles for publication, please contact:
John Pugh & Paul White, Editors, 885 Meadowlands Dr. #509, Ottawa, Ontario, K2C 3N2 Canada; email: streport@objectpeople.on.ca

PRODUCT REVIEWS AND ANNOUNCEMENTS

To submit product reviews or product announcements, please contact the Editors at the address above.

CUSTOMER SERVICE

For customer service in the US, please contact PO Box 5050, Brentwood, TN 37024-5050; 800.361.1279; Fax: 615.370.4845; in the UK, please contact Subscriptions Department, Tower Publishing Services, Tower House, Sovereign Park, Market Harborough, Leicestershire, LE16 9EF, UK; +44.(0)1858.435302; Fax: +44.(0)1858.434958

SIGS BOOKS

For information on any SIGS book, contact: Don Jackson, Director of Books, SIGS Books, Inc., 71 West 23rd Street, New York, NY 10010; 212.242.7447; Fax: 212.242.7574; email: donald_jackson@sigs.com

SIGS CONFERENCES

For information on all SIGS Conferences, please contact: SIGS Conferences,71 West 23rd Street, 3rd Floor, New York, NY 10010; 212.242.7515; Fax: 212.242.7578; email: info@sigs.com

BACK ISSUES

To order back issues, please contact: Back Issue Order Department, SIGS Publications, 71 West 23rd Street, 3rd Floor, New York, NY 10010; Phone: 212.242.7447; Fax: 212.242.7574

REPRINTS

For information on ordering reprints, please contact:
Reprint Management Services, 505 East Airport Road, Box 5363, Lancaster, PA 17601; Phone: 717.560.2001; Fax: 717.560.2063

ADVERTISING

For ad information for any SIGS publication, please contact:

East Coast/Europe:	Gary Portie
Central US:	Elisa Marcus
Recruitment:	Michael Peck
Exhibit Sales, West Coast:	Kristin Viksnins
Exhibit Sales, East Coast:	Sarah Olszewski

Phone: 212.242.7447; Fax: 212.242.7574; email: sales@sigs.com

West Coast:	Diane Fuller

Phone: 408.255.2991; Fax: 408.255.2992; email: dhfsigs@hooked.net

INTERNATIONAL OFFICES

SIGS Conferences Ltd., Brocus House, Parkgate Road, Newdigate, Surrey RH5 5AH, United Kingdom. Phone: 011.44.1.306.631.331; Fax: 011.44.1.306.631.696; email: 100131,3500@compuserve.com.

SIGS France, 105 rue Jules Guesde, 92532 Levallois Perret Cedex, Paris, France. Phone: +33 (1) 41 06 18 00; Fax: +33 (1) 41 06 18 19; email: 100631,1050@compuserve.com.

SIGS Conferences GmbH, Odenthaler Strasses 47, D-51465 Bergisch Gladbach, Germany. Phone: 011.49(0).2202.936.810; Fax: 011.49(0).2202.936.812; email: 100634,2070@compuserve.com.

SIGS HOME PAGE AND ONLINE MAGAZINES

Access the SIGS Home Page at http://www.sigs.com; *Object Currents* at http://www.sigs.com/objectcurrents; and *The X Spot* at http://www.sigs.com/xspot.

```
    (requests includes: 0) ifFalse: [requests grow
                                      replaceAll: nil with: 0].
    requests
        indexOf: 0
        replaceWith: serviceRequestProcess
        startingAt: 1
        stoppingAt: requests size
```

Finally, instances need to be started, stopped, and killed. If you are using ENVY, you'll want to have an application **startUp** method that relays to TcpServer startUp to restart your servers, and a **shutDown** method that relays to TcpServer shutDown to suspend them. Also remember to have a removing method that gets rid of all instances by sending TcpServer initialize.

```
TcpServer:
  resume
    "Begin my server."

    logger == nil ifFalse:
        [logger cr;
            nextPutAll: 'Resuming service on port '; print:
                            self port;
            nextPutAll: ' at '; print: EmTimeStamp now; flush].
    self server resume

  suspend
    "Suspend my server in such a way that when it
     resumes, it opens a new socket. Terminate any
     active requests in process."

    | count |
    logger == nil ifFalse:
        [logger cr;
            nextPutAll: 'Suspending service on port ';
                            print: self port;
            nextPutAll: ' at '; print: EmTimeStamp now.
        count := (requests reject: [:request | request =
                    0]) size.
        count > 0 ifTrue: [logger space; print: count;
                            nextPutAll: ' active requests
                            cancelled.'].
        logger flush].
    requests do: [:request | request = 0 ifFalse: [request
                    terminate]].
    server == nil ifFalse: [server terminate. server := nil].
    socket == nil ifFalse: [socket close. socket := nil]

  terminate
    "Terminate my server and release my state."

    self suspend.
    self class unregister: self.
    logger == nil ifFalse: [logProtect wait. logger close].
    socket := server := service := handler := logger :=
            logProtect := nil

TcpServer class:
  shutDown
    "Suspend all my services so that the image can be
     quit and re-started."

    PortmapControl critical: [Portmap do: [:server |
    server suspend]]

  startUp
    "Re-start all my services."

    PortmapControl critical: [Portmap do: [:server |
    server resume]]!
```

WHAT'S LEFT?

We've run out of space, but this implementation sketch should give you enough "thoughtware" to improvise. We left out the thread–safe log interface (there are problems if multiple processes write to the global Transcript), and our implementation has more extensive logging and security features and a home page facility.

We'll leave you with a handler and a service that implement a complete telnet interface to VisualWorks using this framework, hoping it might inspire your own services.

```
TcpServer class:
  textualHandler
    "Answer a handler that dumps a textual stack."
    "self defaultHandlerFor: 23 is: self textualHandler"

    ^[:exception :stream | stream
        nextPutAll: 'Unhandled exception: '; nextPutAll:
                    exception errorString; cr;
        nextPutAll: exception initialContext printStack]

  evaluationServiceLoop
    "Answer a service block that loops over lines of input,
     evaluating each and sending back the result."
    "self defaultServiceFor: 23 is: self
     evaluationServiceLoop"

    ^[:stream | stream
        next: 6; "Discard garbage characters."
        nextPutAll: 'Smalltalk evaluation service'; cr;
        nextPutAll: 'Type "self close" to end session'; cr;

        nextPutAll: 'Smalltalk> '.
    [stream
        print: (Compiler evaluate: stream nextLine for:
                stream); cr;
        nextPutAll: 'Smalltalk> '] repeat]
```

CONCLUSION

It is easier to write client–server code in VisualWorks than in C, but it is still not easy enough! With some work, you can build a framework that reduces TCP servers to one or two methods.

Next issue, we'll conclude this series by tying together this month's server framework with last month's HTML interface, and you'll have your Smalltalk project documentation on your company's Intranet!