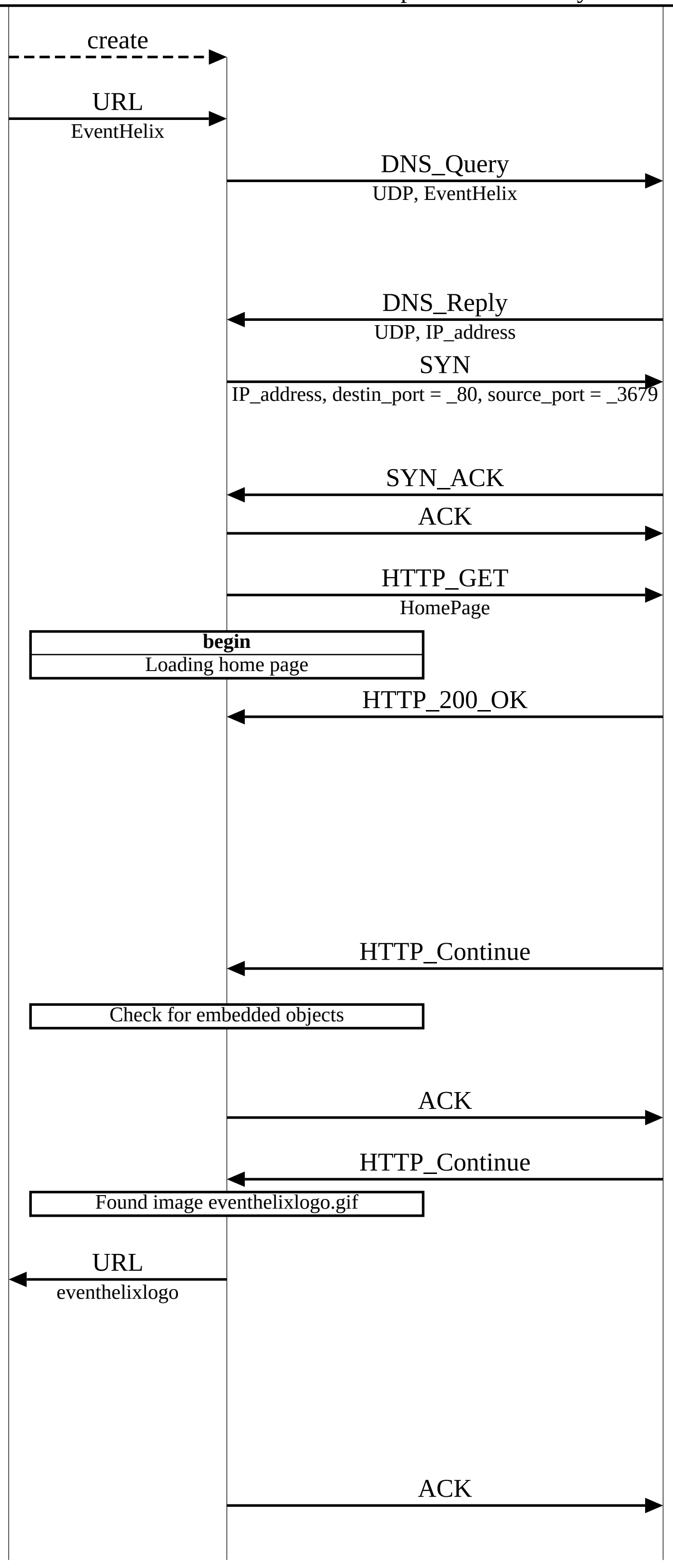
Http_thread_1 Interfaces (Web Browsing)				
client			internet	**EventHelix.com/EventStudio 2.0**
browser			net	
main thread	http thread 1	http thread 2	net	24-Feb-03 07:25 (Page 1)

This sequence diagram describes the IP messages exchanged between the browser and servers on the internet. The message exchange presented here was obtained from an older version of EventHelix.com home page. Internet Explorer (IE) with HTTP 1.1 was used for this message trace.

This is a trace of a real page load and shows all the messages that were involved in rendering the complete page. The actual sequence of packets as seen by the browser is preserved.



Browser creates a new thread to handle the HTTP request

Browser asks the thread to visit www.EventHelix.com

The browser needs to translate from EventHelix.com to an IP address. This is accomplished using the Domain Name System (DNS). A DNS Query message is sent to the DNS Server defined for the PC. The DNS Request is sent as a UDP message

DNS Server translates from EventHelix.com to the IP address and replies back

Browser requests a TCP connection with the web server. The destination port is the well known HTTP port (80). In this case the source port assigned to the socket is 3679.

HTTP server sends SYN+ACK

Three way handshake for TCP connection establishment is complete. The connection is ready for data transfer

Browser sends a HTTP GET for the home page

Web server finds the page and responds with the page. The first segment contains the HTTP header with 200 OK code. This TCP segment also piggybacks an acknowledgement to the HTTP_GET that was sent by the browser.

If the web server was late in sending HTTP GET, the TCP layer on the server would have generated an explicit acknowledgement. See the loading of eventhelixlogo.gif image

Web server sends the second segment containing more page data.

The web browser keeps parsing the partial HTML page, looking for other objects like images that might be needed to render the page.

TCP on the client machine typically sends an ack every two segments

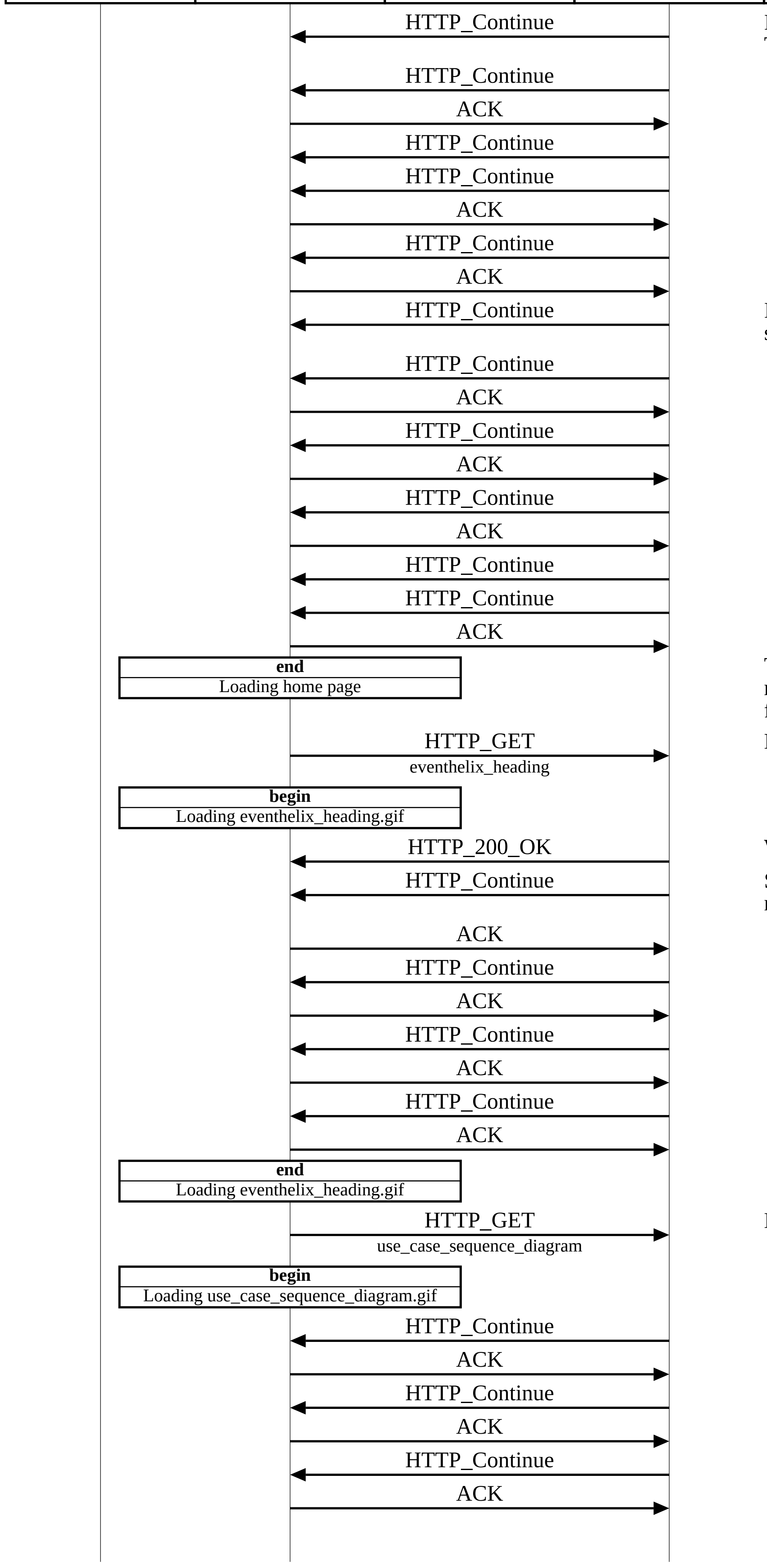
Browser parses the received segment and determines that eventhelixlogo.gif image is needed.

Browser decides that the loading of this image should be handled by a separate thread. Thus the main thread is requested to obtain the URL.

The decision to spawn a thread is based on how much more data is needed to finish loading the current stream. In this case the browser decides that there is quite a bit of data to be loaded from the base page, thus starting another HTTP thread should expedite things.

TCP times out for a second segment and decides to send the ack only after receiving one segment

Http_thread_1 Interfaces (Web Browsing)				
client			internet	**EventHelix.com/EventStudio 2.0**
browser			net	
main thread	http thread 1	http thread 2	net	24-Feb-03 07:25 (Page 2)



In the mean while, the home page thread receives more TCP segments for the page

In the meanwhile, the browser keeps receiving TCP segments for the loading of the main page

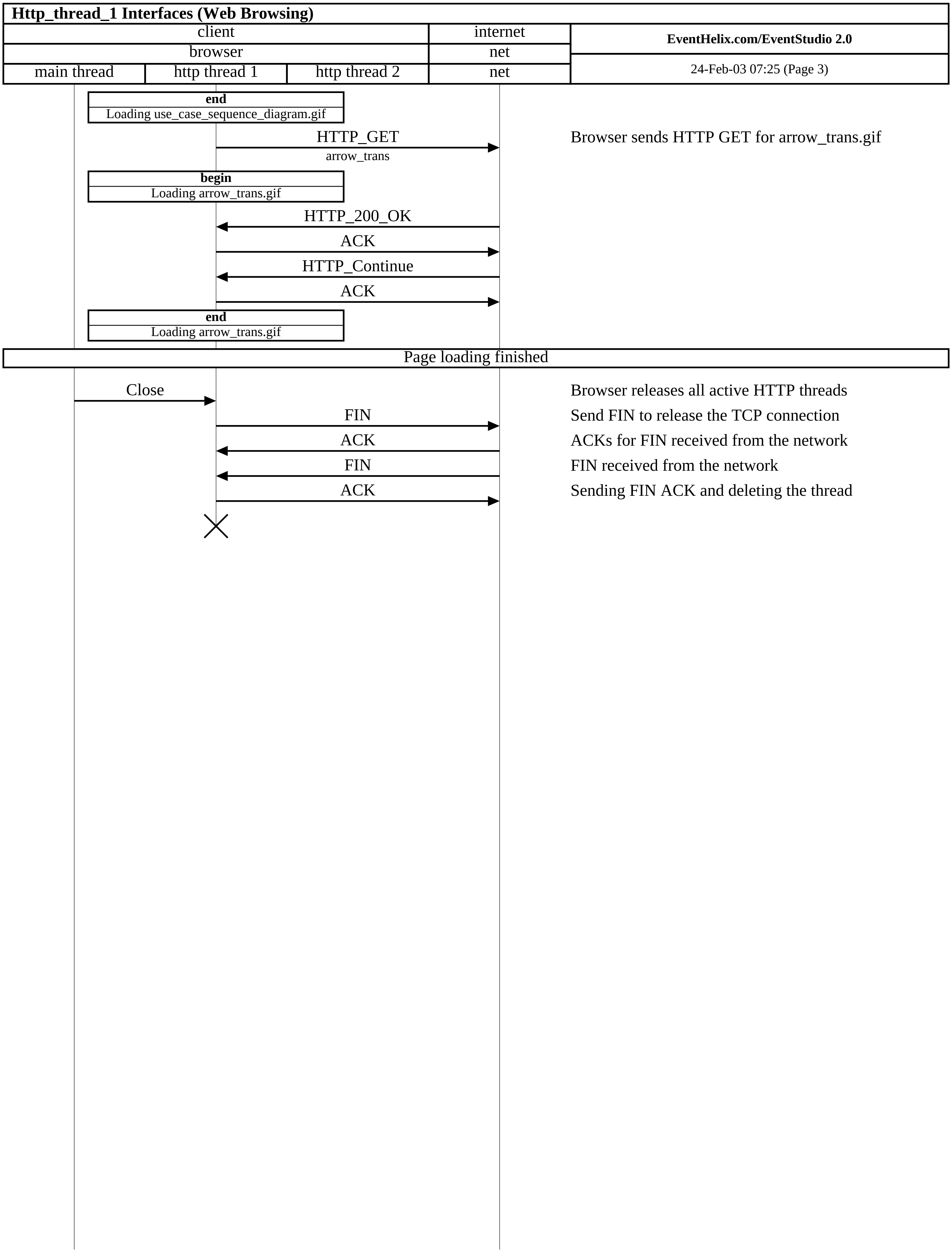
The browser has finished loading of the main page. It reuses the same TCP connection to load pending images from the page.

HTTP GET is sent for eventhelix_heading.gif

Web server responds with HTTP 200 OK

Since this image is large, multiple TCP Segments are needed to load the page

HTTP GET is sent for another image



Http_thread_1 Interfaces (Web Browsing)
client
internet
browser
net
main thread
http thread 1
http thread 2
net
EventHelix.com/EventStudio 2.0
24-Feb-03 07:25 (Page 3)
end
Loading use_case_sequence_diagram.gif
HTTP_GET
arrow_trans
Browser sends HTTP GET for arrow_trans.gif
begin
Loading arrow_trans.gif
HTTP_200_OK
ACK
HTTP_Continue
ACK
end
Loading arrow_trans.gif
Page loading finished
Close
Browser releases all active HTTP threads
FIN
Send FIN to release the TCP connection
ACK
ACKs for FIN received from the network
FIN
FIN received from the network
ACK
Sending FIN ACK and deleting the thread